

AFIT/GCS/ENG/99M-11

A MODELING AND SIMULATION APPROACH TO  
ANALYZE THE WORKLOAD ASSOCIATED WITH THE  
GROWTH OF NETWORK ROUTER ACCESS CONTROL  
LISTS

THESIS

Douglas R. Lomsdalen, Captain, USAF

AFIT/GCS/ENG/99M-11

Approved for public release; distribution unlimited

DTIC QUALITY INSPECTED 2

19990409 089

| REPORT DOCUMENTATION PAGE   |  |   | Form Approved<br>OMB No. 0704-0188 |  |
|---|--|---|------------------------------------|--|
| Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.  |  |   |                                    |  |
| 1. AGENCY USE ONLY (Leave blank)  |  | 2. REPORT DATE<br>March 1999                                    |                                    | 3. REPORT TYPE AND DATES COVERED<br>Master's Thesis                    |
| 4. TITLE AND SUBTITLE<br>A MODELING AND SIMULATION APPROACH TO ANALYZE THE WORKLOAD ASSOCIATED WITH THE GROWTH OF NETWORK ROUTER ACCESS CONTROL LISTS   |  |   |                                    | 5. FUNDING NUMBERS   |
| 6. AUTHOR(S)<br>Douglas R. Lomsdalen, Captain, USAF   |  |   |                                    |  |
| 7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)<br>Air Force Institute of Technology<br>2950 P. Street<br>Wright-Patterson AFB, OH 45433   |  |   |                                    | 8. PERFORMING ORGANIZATION<br>REPORT NUMBER<br><br>AFIT/GCS/ENG/99M-11 |
| 9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)<br>DISA/JEEBC<br>Mr. Ed Cain<br>10701 Parkridge Blvd.<br>Reston, VA 20191-4357<br>DSN: 653-3019 Comm: 1-703-735-3019  |  |   |                                    | 10. SPONSORING/MONITORING<br>AGENCY REPORT NUMBER                      |
| 11. SUPPLEMENTARY NOTES<br>Gregg H. Gunsch, Lt Col, USAF<br>DSN: 785-3636 ext. 4281 Comm: 1-937-255-3636 ext. 4281<br>gregg.gunsch@afit.af.mil  |  |   |                                    |  |
| 12a. DISTRIBUTION AVAILABILITY STATEMENT<br>Approved for public release; distribution unlimited   |  |   |                                    | 12b. DISTRIBUTION CODE   |
| 13. ABSTRACT (Maximum 200 words)<br><p>Organizations can no longer isolate their networks from the rest of the world and still remain competitive. An organization willing to compete in the world market must take the necessary precautions to protect its network, the systems located on those networks, and its mission critical data. There are performance issues associated with the use of access control lists (ACL); however, if ACLs are implemented properly and periodically reviewed, a secure network can be attained.</p> <p>This research attempts to determine how the growth of an ACL affects packet flow and router CPU consumption, and also identify the specific length of an access control list, such that overall router performance is degraded. Additionally, the packet validation model developed for this thesis will be used to provide insights on how access control lists can be optimized.</p> <p>To accomplish the research goals, the ACL Model was built using BONEs Designer. The ACL Model simulated the packet validation component of a network router. Simulations showed packet latency grew linearly as the length of an ACL grows. Optimization efforts showed improvements in the mean packet latency by ordering the ACLs based on a frequency analysis of the incoming data packets and the proper use of ACL terminator entries.</p> |  |   |                                    |  |
| 14. SUBJECT TERMS<br>Access Control List, Extended IP ACL, Reflexive ACL, Dynamic Extended ACL, Network Security, IP Splicing, IP Spoofing, Denial of Service, ACL Performance, Simulation, Designer  |  |   |                                    | 15. NUMBER OF PAGES<br>137   |
|   |  |   |                                    | 16. PRICE CODE   |
| 17. SECURITY CLASSIFICATION<br>OF REPORT<br><br>Unclassified  |  | 18. SECURITY CLASSIFICATION<br>OF THIS PAGE<br><br>Unclassified |                                    | 19. SECURITY CLASSIFICATION<br>OF ABSTRACT<br><br>Unclassified         |
|   |  |   |                                    | 20. LIMITATION OF<br>ABSTRACT<br><br>UL                                |

The views expressed in this thesis are those of the author and do not necessarily reflect the official policy or position of the Department of Defense or the United States Government.

AFIT/GCS/ENG/99M-11

A MODELING AND SIMULATION APPROACH TO ANALYZE THE WORKLOAD  
ASSOCIATED WITH THE GROWTH OF NETWORK ROUTER ACCESS CONTROL LISTS

THESIS

Presented to the Faculty of the Graduate School of Engineering  
of the Air Force Institute of Technology  
In Partial Fulfillment of the  
Requirements for the Degree of  
Master of Science in Computer Systems

Douglas R. Lomsdalen, B.S., M.B.A.

Captain, USAF

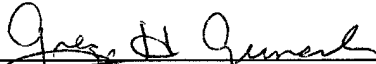
March 1999

Approved for public release, distribution unlimited

A MODELING AND SIMULATION APPROACH TO ANALYZE THE WORKLOAD  
ASSOCIATED WITH THE GROWTH OF NETWORK ROUTER ACCESS CONTROL LISTS

Douglas R. Lomsdalen, B.S., M.B.A.  
Captain, USAF


Approved:

  
\_\_\_\_\_  
Gregg H. Gunsch, Lt Col, USAF  
Chairman

2 MAR 99  
date

  
\_\_\_\_\_  
Stuart C. Kramer, Lt Col, USAF

2 MAR 99  
date

  
\_\_\_\_\_  
Richard A. Raines, Maj, USAF

2 MAR 99  
date

## **Acknowledgments**

This work would not have been possible without the support of several people. I would like to start by saying thank you to Lieutenant Colonel Gregg Gunsch. His guidance and well-rounded knowledge base were beneficial in keeping me motivated and heading in the right direction. His pointed questions and the occasional “So what...?” motivated me to dig deeper into a subject. I would also like to thank my committee members, Lieutenant Colonel Stuart Kramer and Major Richard Raines, for their contribution to my thesis. I also want to extend my thanks to both Ed Cain and Joe Hamlin for their assistance in helping me acquire real world data and access control lists for my simulations. An additional thank you goes to Captain Steve Pratt, his insights in the operation of the modeling tool (BONeS Designer) was beneficial in my successful completion of the model.

Final thanks must go to my family. To my wife, Tatjana, and my two boys, Marcus and Jason: I’ve accomplished a lot over the last year, and it is all due to your constant devotion and support. I couldn’t have accomplished such a large undertaking without your patience, love and understanding. Thank you for being there for me and interjecting words of encouragement, when I couldn’t find them myself.

Douglas R. Lomsdalen

## Table of Contents

|  | Page   |
|--|--------|
| Acknowledgments.....                                     | iii    |
| List of Figures .....                                    | vii    |
| List of Tables .....                                     | x      |
| Abstract .....   | xi     |
| <br>1. Introduction.....                                 | <br>1  |
| 1.1 Motivation .....                                     | 1      |
| 1.2 Background.....                                      | 1      |
| 1.3 Research Problem.....                                | 3      |
| 1.4 Objectives .....                                     | 3      |
| 1.5 Approach .....                                       | 4      |
| 1.5.1 <i>Model Development</i> .....                     | 4      |
| 1.5.2 <i>External Input Data</i> .....                   | 4      |
| 1.5.3 <i>Data Analysis</i> .....                         | 5      |
| 1.6 Scope .....  | 5      |
| 1.7 Thesis Presentation.....                             | 5      |
| <br>2. Literature Review .....                           | <br>7  |
| 2.1 Introduction .....                                   | 7      |
| 2.2 Access Control Lists Defined.....                    | 7      |
| 2.3 Standard/Extended Internet Protocol ACLs .....       | 7      |
| 2.3.1 <i>What are Access Control Lists?</i> .....        | 8      |
| 2.4 Reflexive and Dynamic Access Control Lists.....      | 11     |
| 2.4.1 <i>Reflexive Access Control Lists</i> .....        | 12     |
| 2.4.2 <i>Dynamic Extended Access Control Lists</i> ..... | 14     |
| 2.5 Access Control List Processing Enhancements.....     | 16     |
| 2.6 Network Security .....                               | 17     |
| 2.6.1 <i>A Need for Protective Measures</i> .....        | 17     |
| 2.7 Risks Associated with the Internet .....             | 18     |
| 2.7.1 <i>Loss of Privacy and Data Integrity</i> .....    | 18     |
| 2.7.2 <i>IP Splicing</i> .....                           | 19     |
| 2.7.3 <i>IP Spoofing</i> .....                           | 19     |
| 2.7.4 <i>Denial of Service</i> .....                     | 20     |
| 2.8 Summary.....   | 22     |
| <br>3. Methodology.....                                  | <br>24 |
| 3.1 Introduction .....                                   | 24     |
| 3.2 Requirements of the ACL Model .....                  | 26     |
| 3.2.1 <i>Goals</i> .....                                 | 26     |
| 3.2.2 <i>Simulation Outputs</i> .....                    | 26     |
| 3.2.3 <i>Metrics</i> .....                               | 27     |
| 3.3 Domain .....   | 27     |

|             |  |    |
|-------------|--|----|
| 3.3.1       | <i>Overview</i> .....  | 27 |
| 3.3.2       | <i>System Definition</i> .....   | 28 |
| 3.3.3       | <i>Design of Experiments</i> .....                                     | 29 |
| 3.4         | <i>Issues</i> .....  | 32 |
| 3.4.1       | <i>Input Data Characteristics</i> .....                                | 32 |
| 3.4.2       | <i>Timing Issues</i> .....   | 34 |
| 3.5         | <i>Verification and Validation</i> .....                               | 36 |
| 3.5.1       | <i>Model Verification</i> .....  | 36 |
| 3.5.2       | <i>Model Validation</i> .....  | 37 |
| 3.6         | <i>Summary</i> .....   | 40 |
| 4.          | <i>ACL Model Design and Implementation</i> .....                       | 41 |
| 4.1         | <i>Introduction</i> .....  | 41 |
| 4.2         | <i>Basic ACL Model Construction</i> .....                              | 41 |
| 4.3         | <i>Overview of the ACL Model Parameters</i> .....                      | 42 |
| 4.3.1       | <i>Data Structure Population</i> .....                                 | 42 |
| 4.3.2       | <i>Required Parameters for Simulation</i> .....                        | 46 |
| 4.3.3       | <i>Essential Component Level Variables</i> .....                       | 46 |
| 4.4         | <i>Model Configuration and Data Collection</i> .....                   | 48 |
| 4.4.1       | <i>Input Data</i> .....  | 48 |
| 4.4.2       | <i>Random Number Generation</i> .....                                  | 49 |
| 4.4.3       | <i>Instruction Count Formulation</i> .....                             | 49 |
| 4.4.4       | <i>Data Gathering</i> .....  | 51 |
| 4.5         | <i>Summary</i> .....   | 51 |
| 5.          | <i>Simulation Results and Analysis</i> .....                           | 53 |
| 5.1         | <i>Introduction</i> .....  | 53 |
| 5.2         | <i>Performance Issues Related to the Growth of ACLs</i> .....          | 53 |
| 5.2.1       | <i>Processing Time Grows Linearly</i> .....                            | 53 |
| 5.2.2       | <i>Implications of Findings</i> .....                                  | 59 |
| 5.3         | <i>ACL Optimization Efforts</i> .....                                  | 59 |
| 5.3.1       | <i>Simulation of the AFIT ACL and the Results</i> .....                | 60 |
| 5.3.2       | <i>Simulation of the DISA ACL and the Results</i> .....                | 66 |
| 5.4         | <i>ACL Construction Guidelines to Enhance Router Performance</i> ..... | 70 |
| 5.5         | <i>Summary</i> .....   | 73 |
| 6.          | <i>Conclusion</i> .....  | 75 |
| 6.1         | <i>Introduction</i> .....  | 75 |
| 6.2         | <i>Conclusions</i> .....   | 75 |
| 6.2.1       | <i>Summary of Performance Issues</i> .....                             | 75 |
| 6.2.2       | <i>Summary of the Optimization Efforts</i> .....                       | 76 |
| 6.3         | <i>Possible Model Inadequacies</i> .....                               | 77 |
| 6.4         | <i>Areas of Model Improvement</i> .....                                | 78 |
| 6.5         | <i>Recommendation for Future Research</i> .....                        | 78 |
| 6.6         | <i>Summary Remarks</i> .....   | 79 |
| Appendix A. | <i>Details of IP Access Control Lists</i> .....                        | 80 |
| A.1         | <i>IP Access Control Lists</i> .....                                   | 80 |
| A.2         | <i>Extended IP Access Control List</i> .....                           | 81 |
| A.3         | <i>Summary of Numerical Ranges</i> .....                               | 82 |



|   |            |
|---|------------|
| <i>Appendix B. Samples of Input ACLs and Packet Header Data .....</i> | <i>83</i>  |
| B.1 Single ACL Entry Data.....  | 83         |
| B.2 IP Packet Header Data .....                                       | 83         |
| <i>Appendix C. Access Control List Model .....</i>                    | <i>84</i>  |
| C.1 Access Control List (ACL) Model .....                             | 85         |
| C.2 Instruction Count Variability.....                                | 104        |
| C.3 Summary .....   | 105        |
| <i>Appendix D. Supplemental Graphs .....</i>                          | <i>106</i> |
| Bibliography.....   | 122        |
| Vita.....   | 124        |

## List of Figures

|  | Page |
|--|------|
| Figure 1. Samples from a Numbered Extended IP ACL Entries .....                    | 10   |
| Figure 2. Sample Reflexive ACL for an External Interface .....                     | 14   |
| Figure 3. Sample Lock-and-Key ACL .....  | 15   |
| Figure 4. Expected Behavior of Simulations .....                                   | 30   |
| Figure 5. Sensitivity Analysis Results .....                                       | 39   |
| Figure 6. Comparison of Mean Processing Times for Varying Length AFIT ACL101 ..... | 55   |
| Figure 7. Mean Processing Times for Varying Length AFIT ACL101 .....               | 56   |
| Figure 8. Comparison of Mean Processing Times for Varying Length DISA ACL101.....  | 57   |
| Figure 9. Mean Processing Times for Varying Length DISA ACL101 .....               | 57   |
| Figure 10. Comparison of Mean Processing Times for Varying Length DISA ACL199..... | 58   |
| Figure 11. Mean Processing Times for Varying Length DISA ACL199 .....              | 58   |
| Figure 12. ACL Model Top Level System Module .....                                 | 85   |
| Figure 13. Read ACL File Module .....  | 87   |
| Figure 14. Release of Packets - Loop Module.....                                   | 88   |
| Figure 15. Read Packets - File Module.....   | 89   |
| Figure 16. Read Packets - Random Module.....                                       | 90   |
| Figure 17. While Loop Module .....   | 91   |
| Figure 18. Single ACL from Vector Module.....                                      | 93   |
| Figure 19. Check All Module .....  | 94   |
| Figure 20. Check All & Port Module (TCP/UDP) .....                                 | 95   |
| Figure 21. Check Protocol Module.....  | 96   |
| Figure 22. Check Source Module .....   | 97   |
| Figure 23. Mask Check Module.....  | 98   |

|   |     |
|---|-----|
| Figure 24. Check Destination Module .....   | 100 |
| Figure 25. Check Port Module .....  | 101 |
| Figure 26. Check w/o Port Module (IP/ICMP) .....  | 103 |
| Figure 27. Processing Time (PT) for AFIT Packets Utilizing AFIT ACL - Original .....    | 107 |
| Figure 28. PT for Random Packets (Seed 1) Utilizing AFIT ACL - Original .....           | 107 |
| Figure 29. PT for Random Packets (Seed 2) Utilizing AFIT ACL - Original .....           | 108 |
| Figure 30. PT for Random Packets (Seed 3) Utilizing AFIT ACL - Original .....           | 108 |
| Figure 31. PT for DISA Packets Utilizing AFIT ACL – Original .....                      | 109 |
| Figure 32. PT for AFIT Packets Utilizing AFIT ACL – Optimization Trial .....            | 109 |
| Figure 33. PT for Random Packets (Seed 1) Utilizing AFIT ACL – Optimization Trial ..... | 110 |
| Figure 34. PT for Random Packets (Seed 2) Utilizing AFIT ACL – Optimization Trial ..... | 110 |
| Figure 35. PT for Random Packets (Seed 3) Utilizing AFIT ACL – Optimization Trial ..... | 111 |
| Figure 36. PT for DISA Packets Utilizing AFIT ACL – Optimization Trial .....            | 111 |
| Figure 37. PT for AFIT Packets Utilizing AFIT ACL – Modification .....                  | 112 |
| Figure 38. PT for Random Packets (Seed 1) Utilizing AFIT ACL – Modification .....       | 112 |
| Figure 39. PT for Random Packets (Seed 2) Utilizing AFIT ACL – Modification .....       | 113 |
| Figure 40. PT for Random Packets (Seed 3) Utilizing AFIT ACL – Modification .....       | 113 |
| Figure 41. PT for DISA Packets Utilizing AFIT ACL – Modification .....                  | 114 |
| Figure 42. PT for DISA Packets Utilizing DISA ACL – Original .....                      | 114 |
| Figure 43. PT for Random Packets (Seed 1) Utilizing DISA ACL – Original .....           | 115 |
| Figure 44. PT for Random Packets (Seed 2) Utilizing DISA ACL – Original .....           | 115 |
| Figure 45. PT for Random Packets (Seed 3) Utilizing DISA ACL – Original .....           | 116 |
| Figure 46. PT for AFIT Packets Utilizing DISA ACL – Original .....                      | 116 |
| Figure 47. PT for DISA Packets Utilizing DISA ACL – Optimization Trial .....            | 117 |
| Figure 48. PT for Random Packets (Seed 1) Utilizing DISA ACL – Optimization Trial ..... | 117 |

|   |     |
|---|-----|
| Figure 49. PT for Random Packets (Seed 2) Utilizing DISA ACL – Optimization Trial ..... | 118 |
| Figure 50. PT for Random Packets (Seed 3) Utilizing DISA ACL – Optimization Trial ..... | 118 |
| Figure 51. PT for AFIT Packets Utilizing DISA ACL – Optimization Trial .....            | 119 |
| Figure 52. PT for DISA Packets Utilizing DISA ACL – Modification .....                  | 119 |
| Figure 53. PT for Random Packets (Seed 1) Utilizing DISA ACL – Modification .....       | 120 |
| Figure 54. PT for Random Packets (Seed 2) Utilizing DISA ACL – Modification .....       | 120 |
| Figure 55. PT for Random Packets (Seed 3) Utilizing DISA ACL – Modification .....       | 121 |
| Figure 56. PT for AFIT Packets Utilizing DISA ACL – Modification .....                  | 121 |

## List of Tables

|  | Page |
|--|------|
| Table 1. Deny All While Permitting Authorized Users .....                                | 10   |
| Table 2. Permit All While Denying Malefactors .....                                      | 10   |
| Table 3. Description of Experiments – Baseline ACLs .....                                | 29   |
| Table 4. Description of Experiments – Expanded ACLs .....                                | 31   |
| Table 5. Description of Experiments – Optimized ACLs .....                               | 32   |
| Table 6. Distribution of AFIT Packet Headers by Protocol .....                           | 34   |
| Table 7. Description of Experiments – Sensitivity Analysis .....                         | 38   |
| Table 8. Single ACL Entry Data Structure .....   | 44   |
| Table 9. Packet Header Data Structure .....  | 45   |
| Table 10. Distribution of Packet Headers Categorized by Protocol .....                   | 54   |
| Table 11. Mean Processing Time per Packet ( $\mu$ s) – AFIT ACL Original .....           | 61   |
| Table 12. Mean Processing Time per Packet ( $\mu$ s) - AFIT ACL Optimization Trial ..... | 62   |
| Table 13. Mean Processing Time per Packet ( $\mu$ s) – AFIT ACL Modification Trial ..... | 64   |
| Table 14. Total Processing Times for Input Data by Various ACLs (ms) .....               | 64   |
| Table 15. Mean Processing Time per Packet ( $\mu$ s) – DISA ACL Original .....           | 66   |
| Table 16. Mean Processing Time per Packet ( $\mu$ s) – DISA ACL Optimization Trial ..... | 68   |
| Table 17. Mean Processing Time per Packet ( $\mu$ s) – DISA ACL Modification Trial ..... | 69   |
| Table 18. Total Processing Times for Input Data by Various ACLs (ms) .....               | 70   |
| Table 19. Summary of Numerical Ranges .....  | 82   |
| Table 20. Summary of Modules by Operation .....  | 84   |
| Table 21. Top Level ACL Model Variables .....  | 86   |

## **Abstract**

As the world becomes more interconnected through the use of the Internet, it is imperative organizations take the proper steps to ensure the security of their networks is maintained. Organizations can no longer isolate their networks from the rest of the world and remain competitive. An organization willing to compete in the world market must take the necessary precautions to protect its network, the systems located on those networks, and its mission critical data. There are performance issues associated with the use of access control lists (ACL); however, if ACLs are implemented properly and periodically reviewed, a secure network can be attained.

This research attempts to determine how the growth of an ACL affects packet flow and router CPU consumption, and identify the specific length of an access control list, such that overall router performance is degraded. Additionally, the packet validation model developed for this thesis is used to provide insights into how access control lists can be optimized.

To accomplish the research goals, the ACL Model was built using BONEs Designer. The ACL Model simulated the packet validation component of a network router. Simulations showed packet latency grew linearly as the length of an ACL grows. Optimization efforts showed improvements in the mean packet latency by ordering the ACLs based on a frequency analysis of the incoming data packets and the proper use of ACL terminator entries.

# A MODELING AND SIMULATION APPROACH TO ANALYZE THE WORKLOAD ASSOCIATED WITH THE GROWTH OF NETWORK ROUTER ACCESS CONTROL LISTS

## 1. Introduction

### 1.1 Motivation

As organizations race to make their presence known on the Internet, it is no longer sufficient to rely on physical security (a locked door and safe) to protect their information and networks. Organizations need to look towards high technology solutions to protect their networks, the systems located on those networks, and mission critical data. Without implementing various forms of security, the integrity of an organization's data cannot be assured.

### 1.2 Background

Any organization or person using the Internet is vulnerable to attack by malefactors. An organization's computer network can be attacked in a variety of ways, such as smurfing, IP spoofing, and denial of service attacks. For a network manager, a router is an important and beneficial tool in confining a malevolent attack. By periodically analyzing incoming packets, network activity and anticipating the various types of attacks, the network manager can modify the router's filtering procedures; thereby curbing undesired network activity. Implementing tighter security at the outer fringes of a

network can aid in the overall security of an organization's systems and data. Access control lists are an effective means of controlling the flow of packets through a network.

Access control lists (ACLs) are known by many different names but they can all be reduced to a single objective – protect the systems and data located on an organization's network. Information about how ACLs are implemented is readily available to a certain extent, but published data in regards to router performance figures are unavailable. Various articles have alluded that large access control lists affect the amount of CPU consumption on each interface, but none give any indication as to an exact processing time figure [Har98] [Mor98].

There are many reasons for using access control lists, such as restricting the contents of routing updates, providing efficient traffic flow control, or to decide which types of traffic are forwarded or blocked at the router interface based on an organization's routing policy [Cis98a]. However, the most important reason is the implementation of a security policy. ACLs provide a basic level of security for accessing an internal network; if this step is not done, all packets arriving at the network's border will flow right through. Unchecked, unscrupulous Internet users may send harmful packets into an organization's network. Should a harmful packet be permitted into the network, it may cause the loss of data.

When implementing ACLs, they should be associated with "firewall" routers located at the perimeter of the network being protected. ACLs are not only used to control traffic between internal and external networks; they are also beneficial if employed properly within an internal network. In an internal network, ACLs can be configured to prevent one department from gaining access to another department's information.



### **1.3 Research Problem**

The U.S. Air Force (USAF) and the Department of Defense (DoD) administrative systems are largely dependent on the continued operation of the commercial Internet, and are vulnerable at the weakest link in the communications chain. Coupling the relative ease, low cost and anonymity with which an adversary could mount an attack against the nation's systems, the potential for a degradation of the operational capacity of the DoD administrative systems quickly escalates. To effectively utilize an ACL, it is important to understand how the structure and growth of an ACL can influence the amount of processing accomplished to validate each packet arriving at the router interface. The unavailability of data in identifying the point where router CPU consumption is degraded to an unreasonable point based on the length of the assigned ACL prompted this research effort.

The design portion of this research effort is to model the packet validation process occurring within a network router with an access control list applied. The primary goal is to determine how the growth of an ACL affects packet flow and router CPU consumption, and identify the specific length of an access control list, such that overall router performance is degraded. The secondary goal is to provide insights into how access control lists can be optimized.

### **1.4 Objectives**

The following objectives will be met during this research effort:

- Accomplish a workload study varying the length of network router access control lists used to keep out traffic from untrustworthy sources.
- Identify a universal set of rules to follow when creating an access control list used to optimize network router CPU performance.

To meet the objectives described above, it will be necessary to utilize network-modeling techniques to model a network router and gather information from sources responsible for maintaining router ACLs.

## **1.5 Approach**

The purpose of developing the Access Control List (ACL) Model\* is to simulate the access control list packet validation process occurring within a network router. Real world ACLs and Internet Protocol (IP) packet header data are collected to ensure the model's fidelity while utilizing real world data. After the data is collected, an analysis is accomplished to identify the role the length of an ACL plays on router CPU consumption and how the ACLs can be optimized to increase overall system performance (e.g., minimize mean packet processing times).

### **1.5.1 Model Development**

Using available tools, the ACL Model is designed to simulate the packet validation process occurring within a network router. The simulation tool, BONEs Designer, provides the capability of mimicking the flow of IP packet headers through the assigned ACL. Once the ACL Model is designed, various tests are accomplished in an attempt to validate the model. The test cases use test data and subsets of real world IP packet headers.

### **1.5.2 External Input Data**

Real world ACLs and IP packet header data are collected to ensure the model's fidelity. The IP headers are obtained from the Air Force Institute of Technology (AFIT) and the Defense Information Systems Agency (DISA). Access to this data also affords

---

\* ACL Model refers to the model designed in BONEs Designer for this thesis effort.

the opportunity to accomplish a frequency analysis of the incoming data, providing the information needed to determine how an ACL can be optimized.

### **1.5.3 Data Analysis**

Data is collected throughout the simulation process. Upon completion of the simulations, all data collected is analyzed to determine packet latency and identify the role the length of an ACL plays on router CPU consumption. The results of the ACL optimization efforts for the AFIT and DISA data sets are be examined. Based upon this analysis, a set of guidelines to follow in administering ACLs is provided. A generic set of guidelines will also be created for any system administrator to follow when creating a new ACL; these guidelines are designed to increase overall router performance.

## **1.6 Scope**

In this research, the design and implementation of an ACL Model is accomplished. Important research issues examined are as follows:

- the basic capabilities of the ACL checking facility in a network router; specifically, what processing requirements are needed for Extended IP Access Control Lists,
- the frequency analysis of the real world IP packet header data; specifically, the identification of the addresses most commonly received and the percentage of each type of protocol received,
- the optimization of the access control lists provided by DISA and AFIT, along with the creation of a set of generic techniques to follow in the creation of ACLs , and
- an analysis of the resultant data from the simulations in an attempt to identify a point where an ACL becomes too long and system performance is degraded.

## **1.7 Thesis Presentation**

This thesis is divided into six chapters. Chapter 2 presents background information relevant to this thesis. In particular, it covers what an access control list is and its various

forms. Chapter 3 discusses the methodology used to develop the ACL Model. Chapter 4 describes the design of the ACL Model, including parameters and model operation. Chapter 5 presents an analysis of data collected during the simulations using the ACL Model, specifically identifying performance issues and a set of guidelines to follow in the development of an ACL. Finally, Chapter 6 recaps the efforts and results of this thesis, identifies possible inadequacies of the model, provides areas of model improvement and an avenue for future research.

## **2. Literature Review**

### **2.1 Introduction**

In this literature review, two broad definitions of an access control list are provided. In addition, topics relating to ACLs, such as how ACLs are used and their various forms are discussed. Next, the importance of protecting information systems is discussed and what exactly the systems are being protected from. To conclude this section, a subset of some of the risks associated with being connected to the Internet are also covered, focusing specifically on several forms of attacks designed to disrupt service or cause problems in general for an organization.

### **2.2 Access Control Lists Defined**

Access control lists, also known as ACLs or access lists, can all be decomposed to a single function – protect the data and systems located on an organization's network. ACLs can take on two different forms. The first type of ACLs, not covered in this review, are those associated with operating systems. Within an operating system ACLs are configured to control file and directory access; the lists reflect an organization's desire to restrict access to information based on an individual's or group's need to know. The second form of ACL refers to access control lists associated with network routers, located on the outer edges of an organization's network. The next few sections delve into the specifics of various types of network ACLs and how they compare with one another.

### **2.3 Standard/Extended Internet Protocol ACLs**

In general, access control is an all-encompassing phrase, detailing the mechanisms and policies restricting access to networking resources. Central to most access control

mechanisms are ACLs; they are the basis for determining packet authorization.

Authorization is considered the process by which an individual packet is permitted to gain access to the network; if a packet is denied at the gateway, it is sent to the “bit bucket.”

At the network level, ACLs are used to control the flow of packets into the network. ACLs will not turn a network router into a full-fledged firewall; however, ACLs are an effective tool in controlling network traffic [Mor98].

### 2.3.1 What are Access Control Lists?

There are many different types of access control lists, refer to Appendix A Table 19 for a list of supported protocols and their designated range of numbers. For this research effort, emphasis is placed on numbered Extended IP Access Control Lists; however, this section discusses the differences between standard IP ACLs and Extended IP ACLs.

Access control lists are composed of a group of statements; each statement in the list defines a pattern found in an Internet Protocol (IP) packet or any other type of protocol packet. In order for an ACL to be of any benefit, it must first be loaded into the appropriate router and then assigned to an interface. A router can have multiple interface modules, each of which “connect the router to physical networks (both local and wide area)” [Har98]. Each interface can have one assigned ACL for each type of protocol routed through the interface. For example, if an interface is configured to receive Extended IP and Ethernet packets, two ACLs can be assigned to the applicable interface.

As packets arrive at the router interface, the ACL is checked from top to bottom for a matching entry. The extent of the packet validation process is based upon the type of ACL assigned. If a standard ACL is utilized, an incoming IP packet’s source address is

the only field used for matching purposes (examples of standard IP ACLs can be viewed in Appendix A). Standard ACLs are not designed to look at the incoming packet's session layer protocol. On the other hand Extended IP ACLs, as their name implies, look at a larger set of fields within an IP packet header. Extended IP Access Control Lists use source and destination addresses for matching operations, and session layer protocol information for a finer granularity of control. Finally, if the session layer protocol is of the type TCP or UDP, further filtering can be accomplished by placing controls on the destination port address (see A.2). Overall, the fate of each packet is decided based on an organization's security policy.

Any organization's operating temperament can be gleaned from how it has defined its access control lists. The spectrum of personalities range from the paranoid organization (trust no one) to the trusting. The paranoid company believes the only reason someone would want access to its network would be to cause damage or steal product data; therefore, the paranoid company decides to deny all external access to its network. On the other hand, the trusting organization permits virtually all traffic into its internal network. In practice the trend appears to be a combination of the two styles.

Organizations can decide, based on their security policies, which packets to permit or deny. In the Department of Defense, the mission of each organization determines the filtering policies imposed by the access control lists assigned to its router interfaces.

For the organizations placing themselves in the middle of the spectrum and using a combination of the permit and deny strategy, there are several combinations possible. Two combinations covered here are the following, "Deny all while permitting authorized

users” and “Permit all while denying malefactors.” Tables 1 and 2 identify the pros and cons associated with each policy strategy.

**Table 1. Deny All While Permitting Authorized Users**

| PROS  |
|---|
| <ul style="list-style-type: none"> <li>• This policy is ideal for an organization with a well-defined user set. <ul style="list-style-type: none"> <li>• Overall system security can be maximized under this policy.</li> </ul> </li> </ul>   |
| CONS  |
| <ul style="list-style-type: none"> <li>• ACL maintenance can be difficult and time consuming for an organization with a large number of remote users.</li> <li>• It is difficult to identify all authorized users.</li> <li>• A company attempting to transact E-Commerce over the Internet can’t possibly know all of its customers or potential customers.</li> </ul> |

**Table 2. Permit All While Denying Malefactors**

| PROS   |
|--|
| <ul style="list-style-type: none"> <li>• This policy allows for the widest access to the network. <ul style="list-style-type: none"> <li>• University setting</li> <li>• Customer base</li> </ul> </li> </ul>  |
| CONS   |
| <ul style="list-style-type: none"> <li>• The amount of time required to stay abreast of threats can be extremely high. <ul style="list-style-type: none"> <li>• Based on the amount of activity, the number of identified malefactors can be large; therefore, maintenance of the ACL will be greater.</li> </ul> </li> <li>• An organization needs to identify a reliable source responsible for identifying malefactors (e.g., CERT – Computer Emergency Response Team). <ul style="list-style-type: none"> <li>• It is difficult to prove the identified malefactor is in fact the perpetrator or an innocent bystander who was used by a hacker.</li> </ul> </li> <li>• Inputs from the CERT could arrive too late.</li> </ul> |

In Figure 1, a short access control list is displayed; line one tells the router to deny IP packets with a destination IP address of 109.90.20.13. The second line of the example

```
access-list 101 deny ip 0.0.0.0 255.255.255.255 109.90.20.13 0.0.0.0
access-list 101 permit tcp 0.0.0.0 255.255.255.255 109.90.20.13 0.0.0.0 eq 80
access-list 101 deny ip 0.0.0.0 255.255.255.255 0.0.0.0 255.255.255.255
```

**Figure 1. Samples from a Numbered Extended IP ACL Entries**



permits TCP packets from any address only if the packets are destined for port (80) of host 109.90.20.13. All other IP packets desiring access into the network are denied, per line three. The last line in the configuration (using the deny keyword) can be left off, since IP ACLs implicitly deny all other access; this feature captures all packets (IP, ICMP, TCP and UDP) not explicitly denied or permitted by the ACL. Further examples of Extended IP access lists can be found in Appendix A.

Up to this point ACLs have only been depicted as being used to filter incoming traffic; however, ACLs can also be used to control how packets originating within a network are processed. The need for outbound ACLs can be justified by an organization's security policy; the lists can be designed to prevent employees from wasting company time by incessantly "surfing" the Internet or completing illegal transactions from within the company's network.

According to Peter Morrissey, a network systems programmer at Syracuse University, access control lists **do** exact a toll on router performance. Morrissey also alludes to the fact that a router has to "work harder" as ACLs grow [Mor98]. However, the relationship between the length of an ACL and the performance of a router has not been characterized, or at least published in open sources.

## **2.4 Reflexive and Dynamic Access Control Lists**

Up to this point, the only types of ACLs mentioned were the standard/Extended ACLs. Standard ACLs are relatively easy to work with; their format is always the same (there are only so many ways you can create a legal ACL statement) and the size of the ACL is static. The varying size of the reflexive and dynamic access control lists provides an interesting twist to the standard ACL format. Instead of utilizing numbered ACLs,

reflexive access control lists can only be defined with Extended Named IP Access Control Lists. In Figure 1 the entries depicted a numbered ACL (access-list 101). A reflexive ACL assumes the following form (where the ACL is given a name such as tcptraffic, versus using a numeric designator):

**reflexive IP access list tcptraffic**

#### **2.4.1 Reflexive Access Control Lists**

Reflexive ACLs provide the ability to filter network traffic at a router, based on the packet's session-layer information; in particular, reflexive ACLs are only available for TCP and UDP sessions. Reflexive ACLs are very similar in construct as standard ACLs. Reflexive ACLs contain condition statements defining criteria for permitting IP packets. The processing of a Reflexive ACL is also the same as standard ACLs; the entries are evaluated in order, and when a match occurs, no more entries are evaluated.

There are a couple of notable differences from other types of ACLs. Reflexive ACLs contain only temporary entries; these entries are automatically created when a new TCP or UDP session begins, and the entries are removed when the session ends. Reflexive ACLs are not applied directly to an interface, but are nested within an Extended Named IP Access Control List [Cis98b].

The reflexive entries are generated when a new IP upper-layer (TCP/UDP) session is initiated from within the network [Cis98b]. As a packet from a new session leaves the network, the access control software generates a temporary entry within itself. The temporary ACL entry is designed to permit all traffic associated with the newly created IP session, but will not permit traffic to enter the network if the traffic is not part of the session.

A temporary ACL entry can be terminated in one of three ways for TCP sessions. The entry is removed five seconds after two TCP packets with the FIN bit set are detected (the FIN bit is located in the TCP header). Termination of the session occurs immediately after matching a TCP packet with the RST (reset) bit toggled in the packet header. Or, the entry is removed after a period of inactivity (the time-out period is set by the system administrator).

Reflexive ACLs make spoofing more difficult, since more fields within each packet must be matched before the packet is permitted to pass through the router. In addition, session filtering uses temporary filters, which are removed when a session is over, thus limiting a hacker's window of opportunity.

Figure 2 on the following page depicts a Reflexive ACL. The first seven lines represent entries in an Extended Named IP ACL. Line six is known as the outbound ACL; its primary responsibility is to evaluate all traffic leaving the network via the assigned interface, identifying packets from newly created sessions. Line seven permits all outbound TCP traffic and creates a new ACL named `tcptraffic` (when a new session has been identified). The last two lines in Figure 2 represent the temporary Reflexive ACL and the temporary entry within `tcptraffic`; there can be more than one entry. If there are two or more sessions initiated, each session will have a unique entry in the `tcptraffic` access control list.

1. Before a TCP Session has been initiated.

```
Extended IP access list inboundfilters
permit bgp any any
permit eigrp any any
deny icmp any any
evaluate tcptraffic
Extended IP access list outboundfilters
permit tcp any any reflect tcptraffic
```

2. Added to ACL above after Telnet session initiated.

```
Reflexive IP access list tcptraffic
permit tcp host 172.19.99.67 eq telnet host 192.168.60.185 equal 11005
```

**Figure 2. Sample Reflexive ACL for an External Interface**

### 2.4.2 Dynamic Extended Access Control Lists

Dynamic Extended IP Access Control Lists grant access per user to a specific source or destination host through a user authentication process. In essence, a user can be afforded access through a router dynamically, without compromising security.

#### 2.4.2.1 Lock-and-Key ACLs

The Lock-and-Key is another security feature designed to dynamically filter IP traffic. Similar to reflexive access control lists, Lock-and-Key is configured using IP Extended Named ACLs and can be used in conjunction with other ACL formats. The Lock-and-Key security feature is generally used by an organization wanting to allow a specific remote user or group of users to be able to access the organization's internal network. In this example, remote users Telnet to their organization's network router from IP address 172.18.21.2 (refer to Figure 3). Lock-and-Key automatically attempts to authenticate the user. Upon successful authentication, the user is logged out of the Telnet session and the second statement in the ACL is activated.

```
access-list 101 permit tcp any host 172.18.21.2 eq telnet
access-list 101 dynamic testlist timeout 120 permit ip any any
```

**Figure 3. Sample Lock-and-Key ACL**

Unlike Reflexive ACLs, the dynamic ACL statement is always present in the ACL. All of the packets generated by the user pass through the newly created hole in the router. A flaw of the Lock-and-Key ACL is the fact that, when the remote user is finished with the session and logs out, the hole remains in the router until the ACL entry expires or it is physically removed by the system administrator [Cis98c].

#### *2.4.2.2 Context Based Access Control*

Another example of software dynamically creating access ports within a firewall is Context Based Access Control (CBAC). CBAC is capable of filtering TCP and UDP packets, based on application-layer protocol session information. As long as a session is initiated from within the protected network, all packets related to the established session and arriving from outside of the network are permitted. Similar to Lock-and-Key, CBAC is capable of blocking traffic from sessions that originate outside of the network. When a packet arrives at the router gateway from outside of the network, the packet is inspected to ensure it belongs to an active session [Cis95].

CBAC inspects all outgoing packets to determine if a new session is being initiated. If a new session is initiated, the first outgoing packet triggers CBAC to create a temporary entry in the access control list. The new entry allows returning traffic destined for an active session to pass through the firewall.

## 2.5 Access Control List Processing Enhancements

Due to the compositional differences of standard and Extended IP Access Control Lists, they are processed in a different manner. For example, if a router is configured with a standard IP Access List, the ACL can be cached and the packets take the Fast Switching Path (FSP) [Nyg98]. FSP is a hardware solution, whereby the processing is optimized to provide the fastest flow of packets through the router.

When Extended IP Access Control Lists are applied to an interface, incoming packets flow through the router's CPU [Nyg98]. At Cisco and other companies developing routers, engineers are continuously striving to increase the processing speeds in their routers. The following three process enhancing features (introduced by Cisco) can speed up the flow of packets through a router; however, the enhancements are not being implemented on the AFIT or DISA networks. *NetFlow Switching* can cache the entire session information for each flow of traffic arriving at a router; therefore, only the first packet of a session must be validated against the assigned ACL while the rest of the packets associated with the input stream are allowed to pass through the router. The *Distributed Switching* technique allows the processing to be accomplished by the interface card; therefore, packets are allowed to bypass the CPU. The third enhancement, *Cisco Express Forwarding (CEF)* is used to pre-build forwarding tables. The forwarding table created by CEF contains the same information stored in the *NetFlow* cache; however, it does not require the first packet latency of creating the cache entry. Also, the forwarding table can be downloaded into each interface processor, in effect processing is being accomplished in a distributed fashion [Nyg98].

## **2.6 Network Security**

As network resources and their availability increase, so does the threat that both authorized and unauthorized users will misuse these resources. The privacy and integrity of an organization's entire system is reliant upon security management. The various threats to network access control include spoofing, illegal associations, unauthorized access, denial of service, repudiation and Trojan horses (several of these are discussed in detail later) [Kum97].

This portion of the literature review focuses on the need for strong underlying security architectures. Having an organization's network tied to the Internet makes good business sense; however, there are some recognized threats associated with being hooked to the Internet. There is always the potential of having the network broken into and having company trade secrets stolen. A poorly protected network can quickly become an organization's Achilles' heel.

### **2.6.1 A Need for Protective Measures**

"A recent Ernst & Young survey found that four out of every five large organizations (greater than 2,500 employees) run mission-critical applications on local-area networks (LANs). The LANs and the information they process, increasingly come under direct threat through inter-network connections" [NAT97]. During a National Computer Security Association (NCSA) study of 61 large organizations, NCSA found 142 separate security-breach and system-hacking incidents, all within a 3-month period [NAT97]. The current trend sees companies clamoring to get connected to the Internet before they take the proper steps to protect themselves from an Internet attack.

A recent article in The Sunday Times paints a dismal picture of the United State's frail network infrastructure. A series of "war games conducted by experts has revealed that the world's greatest superpower could be disabled by a handful of determined 'cyber attackers' paralyzing airports, markets and military systems with a few taps on a keyboard" [Cam98]. As America becomes more reliant on technology, the more susceptible we become to a cyber-attack. When played properly, Cyber-warfare can be considered the "balance of power" card. A small country with a grudge against the U.S. doesn't need to spend billions of dollars on offensive weapons, when a couple of million dollars hires a handful of "cyber-mercenaries" capable of penetrating government systems and causing havoc [Cam98]. President Clinton has ordered the coordination of various U.S. counter-terrorism agencies, and the formation of the FBI's new National Infrastructure Protection Center (NIPC), responsible for gauging the vulnerabilities of computer systems to a cyber-attack and finding a way to fight back if the U.S. is attacked [Cam98][CNN98].

## **2.7 Risks Associated with the Internet**

As mentioned earlier, the Internet provides new opportunities for all users; however, the risks involved are ever increasing. Without the proper control mechanisms in place, an organization's data and systems are susceptible to various types of attacks. The next few sections cover issues such as, loss of privacy and data integrity, IP Splicing IP Spoofing, and various Denial of Service (DoS) attacks.

### **2.7.1 Loss of Privacy and Data Integrity**

When two systems are communicating with one another, their electronic dialog is susceptible to packet sniffing. Without using some form of encryption technique, a third



party is afforded the opportunity to eavesdrop. The 1996 Annual Report from the Computer Emergency Response Team Coordination Center listed packet sniffers as one of the most common data intrusion incidents [Cis98d]. Data integrity involves ensuring packets are not tampered with while in transit. Data contained in packets can be afforded some protection if some form of data encryption is utilized.

### 2.7.2 IP Splicing

An attacker using IP splicing can tap into an active session and share the entry point into the network with an authorized user. If the splicing takes place after the authentication phase, the attacker can assume the identity of the already authorized user. To prevent an IP slicing attack, once again an encryption scheme needs to be used at the session or network layer [Ran96].

### 2.7.3 IP Spoofing

In IP source address spoofing, the attacker illegitimately uses a trusted machine's IP address in conjunction with some protocol that authenticates packets based on the packet's IP address [Bel95]. To accomplish this feat, an attacker creates packets with the spoofed source IP addresses. If a firewall is not set up to filter incoming packets whose source address is in the local domain, it can lead to unauthorized remote access to the protected network. Once an intruder gains access to a network, the potential for damage can be immense.

The most effective method of preventing IP spoofing is to configure the access control lists on the inbound router to deny packets with a source address matching those of the internal network [Dae96], and check the source addresses of packets entering the network. It is also important to ensure all packets exiting the network have the internal

network's source IP address. The implication of the source address of an outbound packet being something other than the internal network's IP address points to an individual within the organization that is up to no good.

#### 2.7.4 Denial of Service

Denial of service (DoS) attacks are designed to cause computer systems to slow down considerably or fail. DoS attacks can take many forms and can be targeted towards a single user, a system, or directed against the entire infrastructure. The next several paragraphs discuss various forms of DoS attacks, such as ICMP bombing, Smurfing, Syn/Ack Attack, and the ICMP echo.

##### 2.7.4.1 ICMP Bombing

One use of the Internet Control Message Protocol (ICMP) is to re-route traffic on the fly. For example, one router can broadcast to all other peers that a destination host, a specific IP address, is unavailable. Each interface on a router is identified by a specific IP address; an attacker wishing to stop the flow of traffic to a specific router interface, can accomplish this by sending an ICMP *destination unreachable* command to other routers located on the Internet [Ran96]. In effect, the attacker is telling the world that router X is unavailable and packets should be rerouted or dumped.

##### 2.7.4.2 Smurfing

Smurfing also utilizes the ICM protocol. A perpetrator sends a large quantity of ICMP echo packets to broadcast sites. Each packet sent has a spoofed source address of the victim (target). In the end, the target is inundated with incoming packets from other hosts acknowledging receipt of the ping message. When the packets are delivered to the broadcast sites, each packet is broadcast to a lower network level; on

a multi-access broadcast network there could be hundreds of machines replying to each packet sent.

To prevent a Smurfing attack, it is necessary to have filtering accomplished at the outer edges of the network. An ACL statement such as the following is effective in denying all ICMP packets - **access-list 101 icmp deny any any**; however, an unfortunate side effect of denying all incoming ICMP traffic is that the legitimate uses of this protocol are also affected. In order for an attack of this nature to be effective, the perpetrator is counting on getting the spoofed packets to the "bounce sites" [Hue97]. The broadcast sites can yield an extra ounce of prevention if they turn off directed-broadcasts.

#### *2.7.4.3 Syn/Ack Attack with IP Spoofing*

The syn/ack attack is effective in disabling a site temporarily. A regular network session is initiated through a three-way handshake between a requester and host. The requester sends a *syn* (synchronize) message to a host requesting a session be established. The host responds with a *syn ack* (acknowledgment) back to the requester. The requester completes the transaction by transmitting a final *ack* packet, upon receipt of the final acknowledgment packet the host initiates a session.

The preceding paragraph outlined the steps taken during a legitimate session request. The following steps demonstrate what happens during a syn/ack flood attack. The attacker sends a *syn* message to a host with a spoofed IP address (non-existent). The host responds with a *syn ack*; however, the packet goes to the bit bucket (since the destination address is a phony), thus an *ack* packet never arrives [Bel95]. This action leaves a half-open state at the server. Half-opened states are kept in a queue

until the appropriate *ack* message arrives to complete the handshake. If the above process is repeated a number of times the queue eventually fills up. The problem associated with the syn/ack attack is that legitimate users wishing to establish a session can't, due to the queues being full. To overcome this type of attack, queues need to be flushed periodically or a time-out criterion set for half-open sessions.

#### 2.7.4.4 ICMP Echo

The ICMP echo attack is yet another DoS attack based on the features of the ICMP protocol. Generally ICMP *echo* and *echo reply* messages are sent between routers to request the status of each other. This feature is particularly useful to determine whether a router is active or off-line. Based on the queried router's status, the querying router can direct packets down the appropriate path.

An attacker can use the *echo* commands to get two routers into a looping condition. The attacker initiates an attack by transmitting an echo request to router Y (source IP address is that of router X). Router Y replies to the *echo* packet back to router X. Router X receives the *echo reply*; however, X didn't send an *echo* request, so it interprets the packet as an *echo* request, thus the cycle begins. The circular condition of the ICMP echo attack can quickly affect routing performance.

## 2.8 Summary

Within this chapter different topics have been discussed, each deals with one aspect of network security or another. It is clear that as the proliferation of computers and network systems spread, the risks of individuals using computers for unintended purposes increase. The need for stringent security policies can not be emphasized enough. An organization with an unprotected network is only inviting trouble.

There are many different ways to protect a network. It is important to realize that the security of a network system can no longer be accomplished any more by locking the door to the building as the last person leaves. More aggressive measures need to be taken to shore up the plethora of possible attack options available to a hacker. Security holes provide a hacker a portal into a network, where the hacker can disrupt network activities and possibly corrupt or misuse mission essential data.

The emphasis of this review focused on the use of access control lists as the first line of defense in filtering packets arriving at a network's gateway. There are various forms of ACLs, each with its own special characteristics and functionality. The standard ACL can be utilized as a limited firewall mechanism, while Reflexive and Lock-and-Key access control lists are activated by newly created sessions allowing packets from outside of the network to pass through the firewall. The main purpose of researching access control lists was to find information concerning performance characteristics of ACLs, specifically looking at whether or not the length of an ACL can actually affect router CPU performance. As mentioned in 2.3, the length of an ACL does play a role in router performance; however, no facts or figures to support these claims were found in open source literature.

### **3. Methodology**

#### **3.1 Introduction**

The methodology used for this thesis effort consisted of three phases. The first step focused on the development of the ACL Model, based upon a software solution. The second phase of the research project dealt with acquiring real world Access Control Lists (ACLs) and Internet Protocol (IP) packet headers from the Air Force Institute of Technology (AFIT) and the Defense Information Service Agency (DISA). The final step was the analysis of the data collected from various simulations and the acquired data.

The processing time required to service an incoming packet is continuously diminishing, due to a combination of hardware improvements (e.g., faster CPUs and fast cache) and software features designed to enhance router performance (refer to Section 2.5). However, a poorly organized ACL can have a negative impact on overall router performance. Data is unavailable in identifying the point where the router CPU consumption is degraded to an unreasonable point based on the length of the assigned ACL. One reason for the lack of research in this area can be attributed to the fact that ACLs in the past haven't been very large. However, as threats continue to mount around an organization's network, it is necessary for organizations to take the appropriate steps to protect their networks, hardware, and data. The first step consists of modifying the applicable ACLs to deny network access to known malefactors. Refer to a discussion on the pros and cons of various ACL security policies in Section 2.3.1.

There are three methods to aid in the evaluation of system performance – analytical modeling, simulation, and measurement [Jai91:30]. For this research effort, the

simulation route was explored in the development of the ACL Model. Simulation was selected over analytical modeling and measurement for the following reasons.

- Time required. Setup time of a router specifically configured to test the role the length of an ACL plays on router CPU consumption could have taken too much time and expertise. Designing the ACL Model took a minimal amount of time and the learning curve was small since the tool, BONEs Designer, had been used in the past.
- Tools. The BONEs Designer network simulation tool was readily available, while an idle router was not.
- Accuracy. Analytical modeling provides the lowest level of accuracy, since it requires so many simplifications and assumptions. Simulations require few assumptions and can incorporate more detail, providing a moderate level of accuracy. The accuracy of the measurement evaluation technique ranges from high to none, based on varying environment parameters and system configuration.
- Cost. As mentioned under tools, the simulation tool was already registered and in use, while a router would have to be located, setup and configured. In addition, utilizing a simulation tool allowed for configuration changes with minimal effort.

The ACL Model, a system designed to simulate a network router's packet validation component, uses Designer modules to mimic the flow of packet headers through the ACL located within a router. The role the length of an IP access control list plays on router CPU consumption and how an ACL can be optimized to increase performance are both studied.

In this chapter, the methodology used during the design process of the ACL Model is discussed. The requirement section addresses the objectives of the ACL Model, and how those objectives are met. Also addressed in this chapter is the domain within which this research endeavor lies, explaining why the focus of the research was on Extended IP Access Control Lists. A detailed explanation of the design of experiments is also provided. Various issues relating to the input data's characteristics (the sample ACLs and the IP traffic) and the topic of timing within the simulation is discussed in Section 3.4. The last section concludes the chapter with a discussion on the topic of model verification and validation.

## **3.2 Requirements of the ACL Model**

### **3.2.1 Goals**

The primary goal to be achieved in this endeavor is to show that the length of an access control list does take a toll on router CPU consumption. It is useful to show a link between the length of access control lists and the latency associated between the time a packet begins to pass through the ACL and the time it completes the check. The primary objective of this research is to characterize the point where an ACL becomes so large system performance is degraded beyond an acceptable limit. The secondary objective of this project is to offer some insights into how ACLs can be optimized in regards to complexity, length, structure, and ordering schemes in an attempt to improve system performance.

### **3.2.2 Simulation Outputs**

In a simulation environment, data can be collected at any point deemed necessary in meeting the goals of the simulation endeavor. The output provided by the ACL Model



includes the time required to check each packet against an assigned ACL. In addition, various statistics can be gathered concerning the number of packets permitted and denied by the ACL, along with the total time required to run the simulation.

### **3.2.3 Metrics**

The primary concern of this research effort is to measure the packet latency, in terms of how long it takes the router CPU to match an incoming packet to an entry within the access control list. Also tracked was the total processing time required for a complete set of input data to be scanned against the appropriate ACL; this metric affords the opportunity to contrast the “optimized” ACL to the original ACL in an attempt to develop universal approaches to optimization.

## **3.3 Domain**

### **3.3.1 Overview**

The ACL Model simulates a network router’s packet validation component. Since router manufacturers keep their technology proprietary, it is necessary to identify the processing steps being accomplished within the router. The ACL Model assumes the Extended IP Access Control List packet validation component in a router is implemented in software; [Nyg99] supports this assumption.

Many different types of access control lists can be assigned to a router; however, the focus of the research effort is primarily interested in the processing of IP packets. By narrowing the focus of the research to IP traffic, the domain was reduced to standard and Extended IP ACLs. As mentioned in Section 2.5, packets passing through an interface with a standard IP ACL assigned are processed in hardware. However, packets entering a

router interface with an Extended IP ACL assigned are individually analyzed with a software implementation of the ACL packet validation component [Nyg98].

### 3.3.2 System Definition

The specifics of the ACL Model are discussed in detail in Chapter 4 and Appendix C. However, it is necessary to define the system boundaries established while developing the model, based upon research and tool limitations.

The ACL Model encompasses only the portion of the router responsible for filtering packets based on an assigned ACL. As a packet arrives at a router, the router takes care of routing the packet to the appropriate interface. Each interface in a router can have zero or one ACL assigned. As the packet passes through the interface, the ACL is scanned from top to bottom or until an exit criterion has been met.

To calculate the latency of packets being checked against an ACL, a simulation timer is used. As the simulation starts the process of comparing the header information of an incoming packet against the assigned ACL, the “start time” is stored in the packet’s data structure. The packet header data structure contains pertinent addressing information such as protocol type, source address and port, destination address and port, and two additional fields to store the “start time” and “stop time” for each packet passing through the simulation. Upon completion of the check, the “stop time” is also stored in the packet header data structure. The latency of a packet is simply the difference between the stop and start times. The total time for each collection of packets is also computed, this value is important in the identification of the point where an ACL becomes too large and system performance is degraded. The total time is also used to determine whether or not the “optimized” ACLs perform more efficiently.

### 3.3.3 Design of Experiments

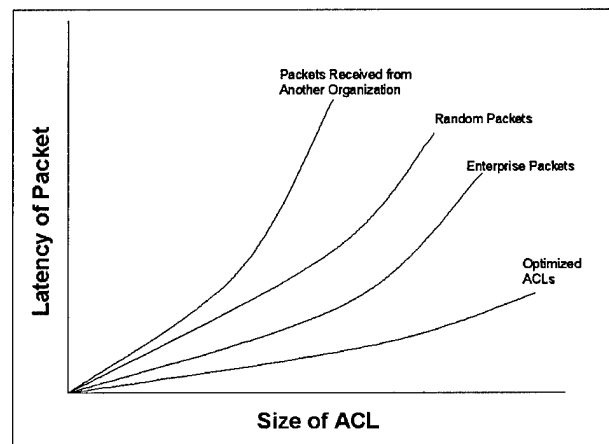
In Tables 3 through 5, the ranges of experiments accomplished during this simulation endeavor are identified. In all experiments 11,443 real world IP packets from AFIT and DISA were used. The initial number of packets provided by DISA determined the number of input IP packets used throughout the simulation process. The first group of experiments in Table 3 scans real world IP packets against their respective ACL; this group is considered the baseline for all other simulations. The second grouping in Table 3 takes 11,443 random IP packets generated by Designer and scans for them in real world ACLs; three different seeds are used for each simulation. Group three in Table 3 looks at the results of taking IP packets destined for one specific router interface (e.g., DISA) and routing them to another interface (e.g., AFIT).

**Table 3. Description of Experiments – Baseline ACLs**

| Group | Description of Experiment                     | Number of Lines |
|-------|---|-----------------|
| 1     | AFIT packets against AFIT ACL 101             | 67              |
|       | DISA packets against DISA ACL 101             | 72              |
|       | DISA packets against DISA ACL 199             | 688             |
| 2     | Random packets against AFIT ACL 101 (3 Seeds) | 67              |
|       | Random packets against DISA ACL 101 (3 Seeds) | 72              |
|       | Random packets against DISA ACL 199 (3 Seeds) | 688             |
| 3     | DISA packets against AFIT ACL 101             | 67              |
|       | AFIT packets against DISA ACL 101             | 72              |
|       | AFIT packets against DISA ACL 199             | 688             |

Figure 4 depicts the expected trend in packet latency as a function of the overall length of an ACL. It is our contention that the longer an access control list becomes the packet latency increases to a point where router performance is hindered. We hypothesize that when incoming packet headers are scanned against the appropriate ACL

(as described in Table 3, Group 1) the packet latency is lower than the instances when random packets or packets destined for another router are applied (Table 3, Groups 2 and 3). It is expected that randomly generated packets have a higher probability of being accepted by an ACL versus packets destined for another router, thereby reducing the overall packet latency. The figure also identifies the expected trend in packet latency for input IP packet headers against an optimized ACL. The knee in each curve represents the point where the length of an ACL adversely affects router performance. The primary goal of this research effort is to identify where the knee occurs, as this is the point where the ACL starts to become too long.



**Figure 4. Expected Behavior of Simulations**

The next nine groups of experiments in Table 4 reflect the same setup as the first three groups, except they are simulated using longer access control lists. The experiments use the same 11,443 IP packet headers as described for Groups 1 through 3. The access control lists were not just doubled, tripled or quadrupled in this effort; caution was used to ensure the terminator entries for the protocols were not duplicated (in the rare cases

where such entries were used). In the event the duplication of a terminator entry should occur, an incoming packet would be prematurely discarded before completely traversing the ACL. Entries designed to capture a particular address were the types of elements increased.

**Table 4. Description of Experiments – Expanded ACLs**

| Group               | Description of Experiment                     | Number of Lines |
|---------------------|---|-----------------|
| ACL Expansion Set 1 |   |                 |
| 4                   | AFIT packets against AFIT ACL 101             | 125             |
|                     | DISA packets against DISA ACL 101             | 138             |
|                     | DISA packets against DISA ACL 199             | 1376            |
| 5                   | Random packets against AFIT ACL 101 (3 Seeds) | 125             |
|                     | Random packets against DISA ACL 101 (3 Seeds) | 138             |
|                     | Random packets against DISA ACL 199 (3 Seeds) | 1376            |
| 6                   | DISA packets against AFIT ACL 101             | 125             |
|                     | AFIT packets against DISA ACL 101             | 138             |
|                     | AFIT packets against DISA ACL 199             | 1376            |
| ACL Expansion Set 2 |   |                 |
| 7                   | AFIT packets against AFIT ACL 101             | 183             |
|                     | DISA packets against DISA ACL 101             | 203             |
|                     | DISA packets against DISA ACL 199             | 2064            |
| 8                   | Random packets against AFIT ACL 101 (3 Seeds) | 183             |
|                     | Random packets against DISA ACL 101 (3 Seeds) | 203             |
|                     | Random packets against DISA ACL 199 (3 Seeds) | 2064            |
| 9                   | DISA packets against AFIT ACL 101             | 183             |
|                     | AFIT packets against DISA ACL 101             | 203             |
|                     | AFIT packets against DISA ACL 199             | 2064            |
| ACL Expansion Set 3 |   |                 |
| 10                  | AFIT packets against AFIT ACL 101             | 241             |
|                     | DISA packets against DISA ACL 101             | 268             |
|                     | DISA packets against DISA ACL 199             | 2752            |
| 11                  | Random packets against AFIT ACL 101 (3 Seeds) | 241             |
|                     | Random packets against DISA ACL 101 (3 Seeds) | 268             |
|                     | Random packets against DISA ACL 199 (3 Seeds) | 2752            |
| 12                  | DISA packets against AFIT ACL 101             | 241             |
|                     | AFIT packets against DISA ACL 101             | 268             |
|                     | AFIT packets against DISA ACL 199             | 2752            |

Groups 13 and 15 in Table 5 detail the various experiments of sending real world IP and randomly generated packets against “optimized” ACLs, based upon a frequency analysis of the incoming IP packet headers and traffic analysis. Each input data set contains 11,443 IP packet headers. Group 14 and 16 detail the experiments where one ACL entry from the “optimized” ACLs is misplaced with the intention of identifying the impact a single entry can have on the overall mean packet latency.

**Table 5. Description of Experiments – Optimized ACLs**

| Group                         | Description of Experiment                              | Number of Lines |
|-------------------------------|--|-----------------|
| AFIT ACL Optimization Attempt |  |                 |
| 13                            | AFIT packets against Reordered AFIT ACL 101            | 70              |
|                               | Random packets against Reordered AFIT ACL101 (3 Seeds) | 70              |
|                               | DISA packets against Reordered AFIT ACL 101            | 70              |
| AFIT ACL Modification         |  |                 |
| 14                            | AFIT packets against Reordered AFIT ACL 101            | 70              |
|                               | Random packets against Reordered AFIT ACL101 (3 Seeds) | 70              |
|                               | DISA packets against Reordered AFIT ACL 101            | 70              |
| DISA ACL Optimization Attempt |  |                 |
| 15                            | DISA packets against Reordered DISA ACL 101            | 75              |
|                               | Random packets against DISA ACL 101 (3 Seeds)          | 75              |
|                               | AFIT packets against DISA ACL 101                      | 75              |
| DISA ACL Modification         |  |                 |
| 16                            | DISA packets against Reordered DISA ACL 101            | 75              |
|                               | Random packets against DISA ACL 101 (3 Seeds)          | 75              |
|                               | AFIT packets against DISA ACL 101                      | 75              |

### 3.4 Issues

#### 3.4.1 Input Data Characteristics

As discussed in the previous section, the first groups of simulations utilized the ACLs in their original form. The last 20 simulations took the same IP packet header data against the “optimized” and modified ACLs. While varying the ACLs in an attempt to

optimize them, all efforts were taken to maintain their completeness and avoid any degradation of their functionality.

Due to the sensitive nature of ACLs, the actual ACLs from DISA and AFIT can not be published in this document. ACLs provide the overall temperament of an organization's security policy, and may inadvertently identify a security hole that may be exploited. In addition, the ACL may identify addresses of known malefactors as identified by a Computer Emergency Response Team (CERT). Announcements made by the Air Force CERT (AFCERT) are not to be released outside of DOD channels. Chapter 5 provides the general security policy of the AFIT and DISA ACLs, without revealing specific addresses.

The general makeup of each ACL is described in Chapter 2 and Appendix A. The ACL data structure used for simulation purposes is discussed in Chapter 4, and the actual format of an input file entry can be viewed in Appendix B.

Initially, the original design of the ACL Model would create random IP packet header data and direct them towards the uploaded real world ACLs; however, further consideration of the potential for an arrival pattern from real world Internet packets identified the need to acquire real data. The sample IP packet headers were collected from routers located at DISA and AFIT; additionally, data was collected from various times of the day at both locations, thus providing a more complete sampling of the various traffic patterns expected throughout an average day. The DISA packets provide a real world representation of the data arriving at the router interfaces housing the ACLs obtained from DISA. The IP packet header data collected from AFIT was captured upon exiting the router; therefore, all of the data had already passed through the assigned ACL.

One could argue that, since the data has already passed through the ACL there could be a large number of packets being denied that are not being accounted for in the frequency analysis used to optimize the AFIT ACL. However, based on the distribution of the AFIT packet headers used in all simulations, we conclude the distribution by protocol would not change significantly by not counting the denied packets. Table 6 depicts the current distribution of packets by protocol; IP packets comprise 94% of all incoming traffic. For example, to alter the choice of the dominant protocol, approximately 11,000 additional TCP packets would have to be denied while at the same time no IP packets could be denied. In terms of the simulation, how does the distribution by protocol relate to the conclusions regarding performance and optimizations? Determining the number of each type of protocol entering the router over a given time period provides system administrators the necessary information to tune their ACL (further information regarding ACL tuning is provided in Chapter 5). As described in the Design of Experiments section, various protocol mixes of packet headers are utilized (through the use of randomly generated packets and packets destined for a different router). Chapter 5 shows how the average packet processing times may be different, but the overall trends remain constant.

**Table 6. Distribution of AFIT Packet Headers by Protocol**

| Input File | Protocol |     |     |      |
|------------|----------|-----|-----|------|
|            | IP       | TCP | UDP | ICMP |
| AFIT       | 10781    | 461 | 176 | 25   |

### 3.4.2 Timing Issues

The ACL Model as originally designed, successfully captured the flow of the packet header information through the packet validation code; however, the timing element was



missing from the model. The Designer tool does not embody the actual router processor; therefore, it was necessary to calculate the processing delays based on the instruction mix of the ACL Model, the router's microprocessor clock-cycles per instruction (CPI), and the cycle period of the router. Due to the proprietary nature and unavailability of the actual code responsible for validating the incoming packets against the assigned ACL, the instruction counts arrived at are based on an estimate of the processes occurring within a router. The CPI value is based upon research in the area of RISC processing accomplished by [EsR91]. The Clock Cycle rate is a simple calculation based on the speed of the processor.

The packet validation process being simulated is designed to mimic the processing accomplished in a Cisco 7500 series router using the MIPS R5000 microprocessor. Research shows the average clock-cycles per instruction (CPI) to be 1.2 [EsR91]. The CPI was instrumental in calculating the CPU time required to complete each block of instructions in the ACL Model. CPU time for each block was calculated in the following manner [HeP94:57]:

$$\text{CPU Time} = \text{Instruction Count Per Block} \times \text{CPI} \times \text{Clock Cycle Time.} \quad (1)$$

Knowing the Cisco 7500 series router has an internal processing speed of 200 MHz on Route Switch Processor 4 (RSP4) [Cis98e], it is possible to calculate the clock cycle time. In this case according to [HeP94:57]:

$$\text{Clock Cycle Time} = 1/(200 \times 10^6) = 5\text{ns.} \quad (2)$$

The clock cycle time formula from above yields a cycle time of 5 nanoseconds. Therefore, access to the instruction count per block in the ACL model, the average CPI for a given microprocessor, and the clock cycle time based on the processor's clock

period rate provided the information needed to apply the appropriate delays in the ACL model. The next chapter delves into the details of the design assumptions regarding timing.

To ensure the packet processing times calculated by the ACL Model are reasonable, meaning the total packet latency as determined by the ACL Model is smaller than the time required to pass through the entire router, the raw packet data received from AFIT and DISA was analyzed. An analysis of the flow data of the packet headers shows the average packet arrival rate is approximately 35  $\mu$ s. Several attempts were made, via correspondence and research, to quantify the average time a packet remains in a router; however, each attempt to obtain this information was unsuccessful.

### **3.5 Verification and Validation**

Routers are utilized extensively throughout the world to transfer electronic packets between points A and B. Each manufacturer of routers designs its hardware and software to operate differently than the competitors. However, a model can be built that simulates the general processes being accomplished in all routers. The specific router process being simulated by the ACL Model is the component responsible for validating packets as they pass through the router's interface; this process is accomplished using access control lists.

#### **3.5.1 Model Verification**

The model verification step involved three phases. Phase 1 of the model verification consisted of using the verification tool included with BONEs Designer. Each module needed to be verified prior to being used in another module or simulation. The types of errors trapped by Designer included: module inconsistencies, construction errors, and dependencies.

The second phase of the verification process involved testing each module in the ACL Model independently. Each module was simulated as a single entity, prior to integrating it into the ACL Model, to ensure the desired output was achieved based on the various test cases.

The final phase of system verification involved attaching the ACL Model modules together and testing the system as a whole. During this phase, the initial tests involved using simple two line access control lists and various IP packet header data files. Designer provides a debug function allowing a user to step through each module in the simulation; this afforded the opportunity to test each viable path the data packets can follow. The outputs of the various tests were within expectations.

### 3.5.2 Model Validation

The creation of a model that is representative of the packet validation process is very challenging, since there are many unknowns and the software is proprietary. Considering this research focuses on a process within a router whose operations are not published, the ACL Model reflects the few known properties, such as microprocessor speed, CPI, and that the packet validation process is accomplished using software. Care was taken in identifying the steps being accomplished within the router based on sound software design principles.

Part of the model validation consisted of accomplishing a sensitivity analysis of the ACL Model. The two sets of simulations relating to the sensitivity analysis can be seen in Table 7. The first set of experiments (Group 1) attempts to quantify the processing time required to validate a packet based on the instruction count values being increased in

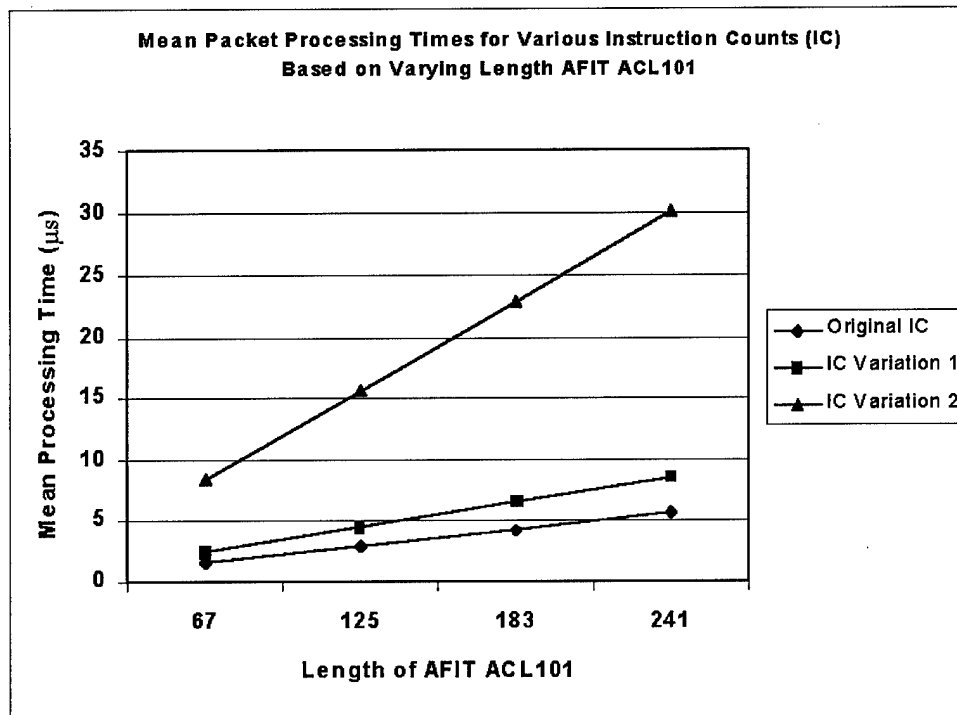
various modules. The instruction count was increased by 14 instructions for the Group 1 simulations. All experiments utilized the AFIT ACL101 and its four varying lengths.

**Table 7. Description of Experiments – Sensitivity Analysis**

| Group | Description of Experiment         | Number of Lines |
|-------|-----------------------------------|-----------------|
|       | Instruction Count Variation One   |                 |
| 1     | AFIT packets against AFIT ACL 101 | 67              |
|       | AFIT packets against AFIT ACL 101 | 125             |
|       | AFIT packets against AFIT ACL 101 | 183             |
|       | AFIT packets against AFIT ACL 101 | 241             |
|       | Instruction Count Variation Two   |                 |
| 2     | AFIT packets against AFIT ACL 101 | 67              |
|       | AFIT packets against AFIT ACL 101 | 125             |
|       | AFIT packets against AFIT ACL 101 | 183             |
|       | AFIT packets against AFIT ACL 101 | 241             |

The second set of simulations (Group 2) answers the question, “What is the impact if the ACL in a router is not stored in registers and each ACL entry must be loaded from memory into the registers?” If this were the case an additional 44 instructions would be required to load a single ACL entry from memory to the appropriate registers.

The results of the simulations were as expected; the processing time for the various simulations increased due to the increase in instruction count values. The bottom line is the growth in mean processing time remains linear despite the increase in instruction counts values. The results of the sensitivity analysis can be seen in Figure 5. The figure depicts the results of the two simulation sets described above compared to the results of the AFIT packets being scanned by the ACL Model with the original instruction count values (refer to Table 21 in Appendix C for the original instruction count values).



**Figure 5. Sensitivity Analysis Results**

Wherever possible the design of the ACL Model is based upon available data concerning the operation of a router. The overall design of the ACL Model reflects the inputs of my peers and the insights provided by my advisor. The results of the modeling and simulation approach are only as valid as the logic and parameters used to construct the model. In an attempt to further validate the ACL Model, copies of the model were sent to personnel at Cisco and DISA. A representative from Cisco offered the following, “Your model is reasonably accurate for the configuration you have given” [Nyg99]. To aid in the statistical relevance of the output, multiple simulation runs will be performed with various global seeds, varying lengths of access control lists, and IP packet headers collected from various times of the day.

### **3.6 Summary**

This chapter presents a high-level discussion of the approach chosen for this thesis. Effort was concentrated on identifying what the ACL Model is capable of accomplishing and how the model utilized the input data in the form of real world access control lists and IP packet headers. Also covered was the domain of the simulation, to include a description of the system definition and the design of experiments. The final sections discussed the steps taken to verify and validate the ACL Model.

Chapter 4 addresses the top level system design of the ACL Model and how it simulates the ACL checking facility located in a router. Also addressed is an explanation of the various data structures, memory modules and parameters.

## **4. ACL Model Design and Implementation**

### **4.1 Introduction**

The Access Control List (ACL) Model was designed to simulate the flow of IP packet headers through a router interface with an assigned ACL. The model environment was designed so that variations to the ACLs could be introduced and subsequent measurements taken to evaluate system performance. This chapter captures the essence of the many design decisions made during the creation of the ACL Model and discusses their validity as a representation of the IP packet validation process occurring within a router. This chapter concludes with a discussion on the topics of model configuration and data collection.

### **4.2 Basic ACL Model Construction**

The ACL Model was developed using BONEs Designer. The approach taken in the development of the ACL Model was that of small incremental steps, building from the bottom up. Each level of the model was built using modules designed specifically for the ACL Model. Appendix C is designed to take the reader through the simulation start up, initialization and all of the packet validation steps. The appendix contains a detailed explanation of each module and top-level system parameters. Appendix C is designed to be a supplement to this chapter.

In various publications from Cisco, one of the major manufacturers of network routers, the authors freely use the term “software” when referring to access control. The ACL Model builds on the fact that the Extended IP Access Control List scans are accomplished using software [Cis97].

The ACL Model does not embody an entire router, or even the interface on a router. The sole purpose of the model is to accomplish the packet validation process. It is assumed the interface has forwarded the packets to the ACL. In addition, the focus of this thesis effort deals primarily with routers at the outer edge of a network, their primary function being to scan each packet header to determine whether or not the packet should be allowed to pass through the network.

### **4.3 Overview of the ACL Model Parameters**

In the previous section, the underlying assumptions complied with while designing the ACL Model were identified. This section focuses on various links between the different modules in the ACL Model, in terms of their parameters and variables.

The creation of the ACL Model involved the use of many different variables and parameters; at this juncture it is necessary to identify the most important parameters of the ACL Model. A complete listing of all parameters and variables can be found in Appendix C, Table 21.

#### **4.3.1 Data Structure Population**

A data structure is defined to meet the needs of the model and does not need to duplicate all of the functionality of a system being simulated. During the simulation in Designer, data structures travel through the system triggering events and collecting data. The data employed by the ACL Model is passed from one module to another via two data structures, *Single ACL Entry* and *Packet Header*, designed exclusively for this modeling effort. The standard libraries accompanying the Designer modeling package included a predefined data structure similar in construct to the *Packet Header*; however, the data structure contained excess fields not necessary for the ACL Model. Therefore, following



good modeling practices, the *Packet Header* data structure was designed to contain only the fields necessary for the simulations.

#### **4.3.1.1 Single ACL Entry**

At the commencement of the simulation, the entire access control list is placed into a vector identified as *ACL in Memory*. Once the first packet is “received from the router interface” the *Single ACL Entry* data structure (Table 8) is populated by values from the *ACL in Memory* vector. The *ACL Vector Counter* maintains the value of the starting position of the next ACL entry in the vector for each iteration of the *While Loop* module (Figure 17 in Appendix C).

**Table 8. Single ACL Entry Data Structure**

| <b>Field Name</b>   | <b>Description</b>   | <b>Legal Range</b> |
|---------------------|--|--------------------|
| Action              | This field identifies what action (permit or deny) is to occur when an incoming packet matches the ACL entry   | [0,1]              |
| Type                | This field identifies the type of packet the ACL entry is designed to filter. In this thesis there are four types of protocols dealt with: IP (0), ICMP (1), TCP (2), and UDP (3).   | [0,3]              |
| ACL Source Dot1     | The addressing scheme of Internet Protocol Version 4 involves a 32-bit address, divided into 8 bit pieces. ACL Source Dot1 represents the first 8 bits of the source address.  | [0,255]            |
| ACL Source Dot2     | ACL Source Dot2 represents the second 8-bit sequence of the source address.  | [0,255]            |
| ACL Source Dot3     | ACL Source Dot3 represents the third 8-bit sequence of the source address.   | [0,255]            |
| ACL Source Dot4     | ACL Source Dot4 represents the fourth 8-bit sequence of the source address.  | [0,255]            |
| ACL SourceMask Dot1 | ACL SourceMask Dot1 represents the mask to apply to Source Dot1  | [0,255]            |
| ACL SourceMask Dot2 | Mask to apply to Source Dot2   | [0,255]            |
| ACL SourceMask Dot3 | Mask to apply to Source Dot3   | [0,255]            |
| ACL SourceMask Dot4 | Mask to apply to Source Dot4   | [0,255]            |
| ACL Dest Dot1       | Similar to the ACL Source Dot1 description, ACL Dest Dot1 represents the first 8 bits of the destination address.  | [0,255]            |
| ACL Dest Dot2       | ACL Dest Dot2 represents the second 8-bit sequence of the destination address.   | [0,255]            |
| ACL Dest Dot3       | ACL Dest Dot3 represents the third 8-bit sequence of the destination address.  | [0,255]            |
| ACL Dest Dot4       | ACL Dest Dot4 represents the fourth 8-bit sequence of the destination address.   | [0,255]            |
| ACL DestMask Dot1   | ACL DestMask Dot 1 represents the mask to apply to Destination Dot1  | [0,255]            |
| ACL DestMask Dot2   | Mask to apply to Destination Dot2  | [0,255]            |
| ACL DestMask Dot3   | Mask to apply to Destination Dot3  | [0,255]            |
| ACL DestMask Dot4   | Mask to apply to Destination Dot4  | [0,255]            |
| Operation           | When the incoming packet protocol is of the type TCP or UDP there is an operation assigned, else this value is 0. In the Extended IP ACL there are four types of operations: Greater Than (1), Less Than (2), Equal (3) and Not Equal (4). | [0,4]              |
| Port Number         | When the incoming packet protocol is of the type TCP or UDP this field contains the destination port to be compared to.  | [0,65535]          |

#### 4.3.1.2 Packet Header

Once the entire ACL has been loaded into memory, the data file containing the Internet Protocol (IP) packet header data is opened. Once the file is opened, the *File Mem Pkt* pointer is utilized to identify where the next IP packet starts. The *Packet Header* data structure (Table 9) is filled by extracting the values from the external file. Only one packet header is read from the file at a time.

**Table 9. Packet Header Data Structure**

| <b>Name</b>      | <b>Description</b>  | <b>Legal Range</b> |
|------------------|---|--------------------|
| Protocol         | This field indicates which upper layer protocol is to receive the data portion of the datagram [Dav88].   | [0,3]              |
| Source Dot1      | The addressing scheme of Internet Protocol Version 4 involves a 32-bit address, broken up into 8 bit pieces. Source Dot1 represents the first 8 bits of the source address.           | [0,255]            |
| Source Dot2      | Represents the second 8-bit sequence of the source address.   | [0,255]            |
| Source Dot3      | Represents the third 8-bit sequence of the source address.  | [0,255]            |
| Source Dot4      | Represents the fourth 8-bit sequence of the source address.   | [0,255]            |
| Destination Dot1 | The addressing scheme of Internet Protocol Version 4 involves a 32-bit address, broken up into 8 bit pieces. Destination Dot1 represents the first 8 bits of the destination address. | [0,255]            |
| Destination Dot2 | Represents the second 8-bit sequence of the destination address.  | [0,255]            |
| Destination Dot3 | Represents the third 8-bit sequence of the destination address.   | [0,255]            |
| Destination Dot4 | Represents the fourth 8-bit sequence of the destination address.  | [0,255]            |
| Source Port      | If the protocol is of the type TCP or UDP, there is a source port address assigned; else the value in this field is 0.  | [0,65535]          |
| Destination Port | If the protocol is of the type TCP or UDP, there is a destination port address assigned; else the value in this field is 0.   | [0,65535]          |
| ACL Start Time   | The time a packet begins traversing through the ACL is stored in this field.  | [0,∞]              |
| ACL Stop Time    | The time a packet stops traversing through the ACL is stored in this field.   | [0,∞]              |

### 4.3.2 Required Parameters for Simulation

There are many configurable parameters in the ACL Model; however, there are four parameters in particular an individual using the system must properly set to have a successful simulation. The parameters identify the two files to open and the number of entries in each file.

#### *4.3.2.1 ACL File to Open and Packet Header File to Open*

The user of the ACL Model must identify the location and names of the two input files necessary for the operation of the model. As mentioned before and shown in Appendix B, the precise format of the files is paramount in the overall success of the simulation.

#### *4.3.2.2 Number of Lines in ACL and Number of Packet Headers*

In order for the ACL Model to accurately mimic the operations occurring in a network router, an accurate line count of each input file is important. The two line count values are used exclusively in controlling the two main looping mechanisms within the model. In the event an input value is too large, errors to the effect, “exceeded end of file” are encountered. If the input numbers are too small, the entire ACL will not be loaded into memory and all of the packet headers will not be tested; in the end, the probability of a skewed resultant data set is increased.

### 4.3.3 Essential Component Level Variables

The previous section identified the four essential parameters required for a successful simulation. This section looks at several variables required to control the flow of the simulation. The Designer tool took care of assigning the appropriate variables for the purpose of controlling the loop modules. The next few sub-sections highlight the two

variables used specifically to control the flow of the simulation. Also explained are two variables used to track the final status of each incoming packet.

#### **4.3.3.1 Found Flag**

As the name implies, *Found Flag* is simply an integer variable initialized to zero and incremented to one (1) if the incoming packet header has passed all of the checks associated with an entry in the applicable ACL. In the case where the *Found Flag* is greater than zero, the *While Loop* module is exited, all counters and flags are reinitialized, and the next incoming IP packet header is retrieved from the input file.

#### **4.3.3.2 Failed Check**

Similar to the *Found Flag*, the *Failed Check* variable is also a flag. When the *Failed Check* variable is tripped (assigned a value greater than zero), the program returns control to the *While Loop* module, where the program acquires the next ACL entry or exits the loop and retrieves the next IP packet header. The *Failed Check* flag is located in each Designer module where a comparison is made between the entries in the *Packet Header* and the *Single ACL Entry* data structures.

#### **4.3.3.3 Number of Packets Permitted and Number of Packets Denied**

These two variables do not have any bearing on the flow of the simulation, they were designed to identify the number of packets being permitted or denied access to the router. In the majority of the simulations, the final value of each variable was inconsequential. However, in the simulations concerned with testing the ACL optimization techniques, the values in the two variables provided a means of identifying whether or not all of the packets were being captured by the ACL terminator entries.

## **4.4 Model Configuration and Data Collection**

The primary thrust of this research effort was to identify the point where the length of an ACL starts taking a toll on the overall system performance of the network router upon which it is utilized. Also studied were ways to optimize the ACLs, by modifying the order of the entries and noting the differences in processing times. Baselines of the model are required so subsequent simulation data can be compared and meaningful data extracted.

Table 3 identifies the set of experiments considered the baseline for this research effort. Baseline values consist of the mean processing times for an individual packet header to be scanned against the original versions of the ACLs. For example, the values in Table 11 represent the baseline values for the AFIT ACL101.

The following sections cover the test conditions common to all runs. The topics covered include an overview of the input data, a discussion on how the same random IP packet headers are generated for each simulation, a description of the assumptions followed when identifying the instruction count for each module, and how the data is collected and analyzed.

### **4.4.1 Input Data**

The input IP packet files and randomly generated packets were standard in length, composition and content throughout all simulations. In addition, within this model no packets were lost due to queue overflows. Each simulation tested all 11,443 sample incoming IP packet headers against various ACLs.

#### 4.4.2 Random Number Generation

The ACL Model contains several random number generators used to create random IP packet headers. Within the BONEs Designer environment, each generator block is supplied with a unique large integer to act as a seed. However, if the number supplied to each generator is a -1, the global seed is used, allowing identical simulations to be run. The random number generators in the *Read Packets – Random* module (Figure 16 in Appendix C) all use -1 as their seed, allowing the global seed to control the simulation.

#### 4.4.3 Instruction Count Formulation

To accomplish the task of assigning a time value to each module in the ACL Model, it was necessary to complete an in-depth review of each module and identify the instruction set most likely used for packet validation in a router. The following paragraphs discuss some of the assumptions made in terms of the ACL Model and the computation of the delay for each module. Appendix C identifies the instructions used to determine the instruction count for each module.

##### 4.4.3.1 ACL Model Assumptions

Several assumptions were made when calculating the instruction count for each of the major components of the ACL Model. Instruction counts were not calculated for every step in the model, only those operations contained in the software responsible for scanning the incoming packet headers against the assigned ACL. In the event instructions are overlooked, the instruction counts would need to be increased. Based on the sensitivity analysis described in Chapter 3, it is expected that increasing the instruction count can increase the overall mean packet latency induced by the packet validation component. Figure 5 depicts the increase in mean processing time per packet while being

scanned against the AFIT ACL101 of different lengths. The results of the sensitivity analysis showed the growth of the mean processing time to be linear even in the event the instruction counts must be increased.

- The ACL Model attempts to simulate the packet validation process associated with a Cisco 7500 series router. Since this series of routers uses the MIPS RISC chip, all instructions use the MIPS assembly language instruction set.
- When an access control list is applied to an interface on a router, it was assumed the entire ACL is stored in registers. The program maintains a mapping of each element and knows where and when each register is used. In the ACL Model, the series of registers are represented as the *Single ACL Entry* data structure. This assumption was also tested during the sensitivity analysis accomplished in Section 3.5.2.
- When the instruction **jump** was used, a deviation was made from the normal syntax. Generally, the MIPS **jump** instruction requires an address location to jump to; in this case the module is identified instead of an address. The same assumption applies to the **beq** (branch on equal) instruction.
- When a new Internet Protocol packet arrives at the router interface it is immediately placed in a series of registers, represented in the ACL model as the *Packet Header* data structure.
- When identifying the instructions, the focus was on identifying the appropriate instruction to be executed by the CPU versus identifying each and every register and memory location being used.



#### 4.4.3.2 Delay Computation

Throughout the simulation, absolute delay blocks were inserted to increment the CPU processor clock. The delay for each block was calculated based on the CPI of the RISC processor, the Clock Cycle time, and the applicable Instruction Count for each module. As an example of the delay computation, assume an incoming packet must accomplish a *Mask Check*; the delay associated with each *Mask Check* is  $2.4 \times 10^{-8}$  seconds (24 nanoseconds). The delay of 24 nanoseconds is based on the *Mask Check* module's instruction count of four (4), a CPI of 1.2, and a Clock Cycle rate of 5ns.

#### 4.4.4 Data Gathering

Once the instruction count figures have been calculated and input into the model, the model can be run to determine processing times. Once all of the simulations are run, as discussed in Chapter 3, various factors can be observed to note trends in processing times: ACL Start and Stop times, and the final counts of the number of packets permitted or denied.

Finally, all data collected is analyzed in an attempt to identify the point where the packet latency is adversely affected due to the length of the applied ACL. Another byproduct of the simulations relating to the optimization efforts is a set of recommendations identifying ways to decrease packet latency.

### 4.5 Summary

This chapter provided a top level description of the ACL Model architecture; a model designed and used to simulate the processing accomplished within a network router while scanning an incoming IP packet against an assigned ACL. The primary parameters and variables responsible for the overall operation of the ACL were discussed. The details of

the ACL Model are included in Appendix C. Efforts were taken to mimic the logical flow of packet headers through the network router packet validation phase. Also addressed was the topic of determining the instruction count in order to calculate the appropriate module delays. Chapter 5 discusses the analysis of the simulation data and highlights ACL optimization techniques.

## **5. Simulation Results and Analysis**

### **5.1 Introduction**

This chapter presents an analysis of data collected during the simulations using the ACL Model. During testing, data was collected to meet two objectives identified in Chapter 1. The results for each objective are presented separately, starting with a discussion of the performance issues related to the growth of ACLs. The final section of this chapter embodies a set of universally applicable guidelines to follow when developing or modifying an access control list, to ensure incoming packets are processed in the most efficient manner.

### **5.2 Performance Issues Related to the Growth of ACLs**

Throughout this thesis it has been conveyed that the larger an ACL becomes, the longer the network router's CPU must work to determine the final status of an incoming packet (packet latency). The ACL Model showed the previous statement to be true. However, the various simulations were inconclusive, in identifying a point (the "knee" as depicted in Figure 4) where the router's performance was significantly degraded.

#### **5.2.1 Processing Time Grows Linearly**

The ACL Model identifies how long it takes each packet header in the input data set to pass through the packet validation phase of the router. However, the model falls short in identifying a breaking point in system performance. Three different access control lists are utilized during this research; and as discussed in Chapter 3, each list increases in overall size to simulate growing ACLs.

As explained in Section 3.3.3, in addition to three randomly generated sets of IP packet headers, two input files were applied against each access control list (to include

the original and all expanded forms of the ACL). The first input file contained real world packet headers from AFIT while the second file included packet headers from DISA. Table 10 categorizes the contents of the input data, based on the packet header's protocol type.

**Table 10. Distribution of Packet Headers Categorized by Protocol**

| Input Data    | Protocol |      |      |      |
|---------------|----------|------|------|------|
|               | IP       | ICMP | TCP  | UDP  |
| AFIT Packets  | 10781    | 25   | 461  | 176  |
| Random Seed 1 | 2931     | 2890 | 2780 | 2842 |
| Random Seed 2 | 2959     | 2847 | 2817 | 2820 |
| Random Seed 3 | 2827     | 2846 | 2934 | 2836 |
| DISA Packets  | 0        | 173  | 8860 | 2410 |

Each experiment yielded a list of values representing the processing time required for each packet header to traverse the ACL. The mean processing time was computed for each set. Figure 6 provides a comparison of the various mean processing times associated with packet headers scanned against varying lengths of the AFIT ACL. In lieu of plotting three separate lines representing the random data sets, the depicted value represents the average value of the three random data set means. For example, for the AFIT ACL of length 67, the mean processing times associated with the random data sets were 1.525, 1.527, and 1.535 microseconds ( $\mu$ s); the average value of the means was 1.53  $\mu$ s. The motivation behind combining the three mean values of the random data sets was an attempt to ensure the graph remained legible.

Figures 6 and 7 represent the results of the various experiments accomplished using the ACL Model with the AFIT access control list. In each case, the results show a linear growth in the mean processing time required for a packet header to pass through the varying length ACLs. In general, most of the incoming packets pass through the entire

ACL; therefore, we expect the observed linear growth of the packet latency to be worst case. Sublinear growth in packet latency can be expected by simply fine tuning the ACL. Fine tuning the ACL involves following the optimization guidelines outlined in Section 5.4. In nearly all cases, the rise over run of the mean processing values is within 10 nanoseconds (ns) of each other (less than 1% difference). Comparing the 24 ns it takes to complete a single *Mask Check* (discussed in Section 4.4.3.2) to the difference in the rise over run value, 10 ns is equivalent to approximately two ACL Model instructions. In relation to the mean processing time for a single packet, 10 ns represents approximately 0.6 percent of the total time.

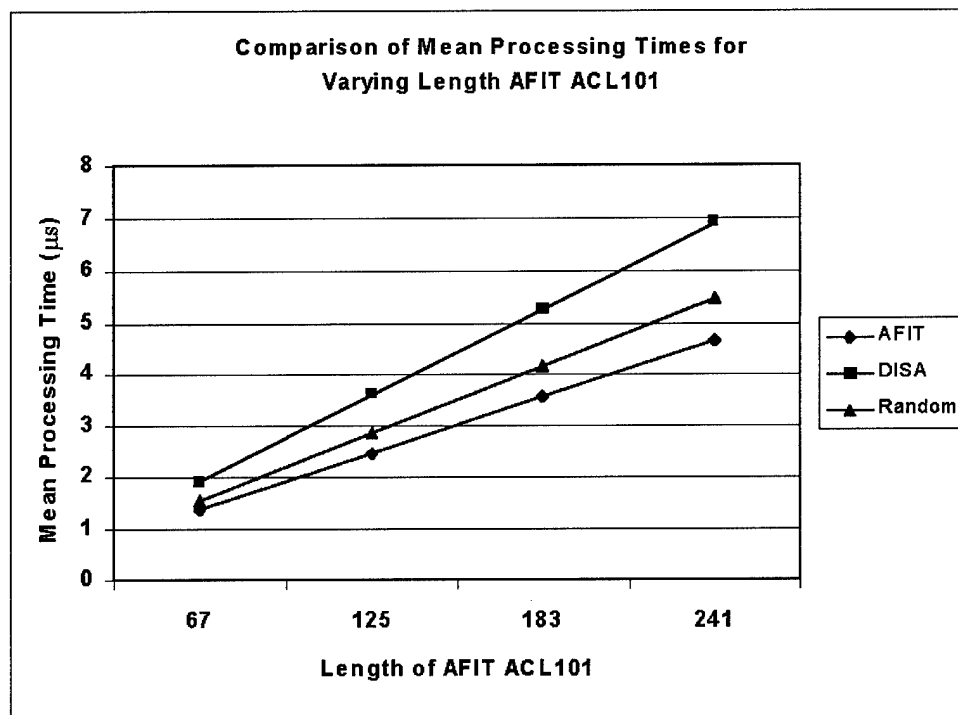
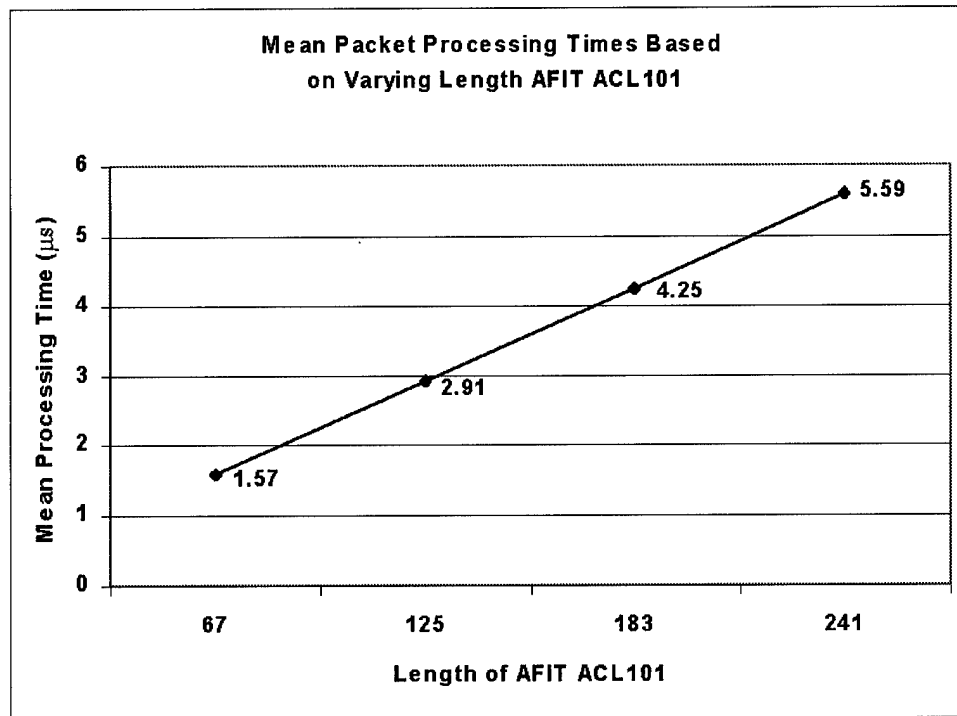


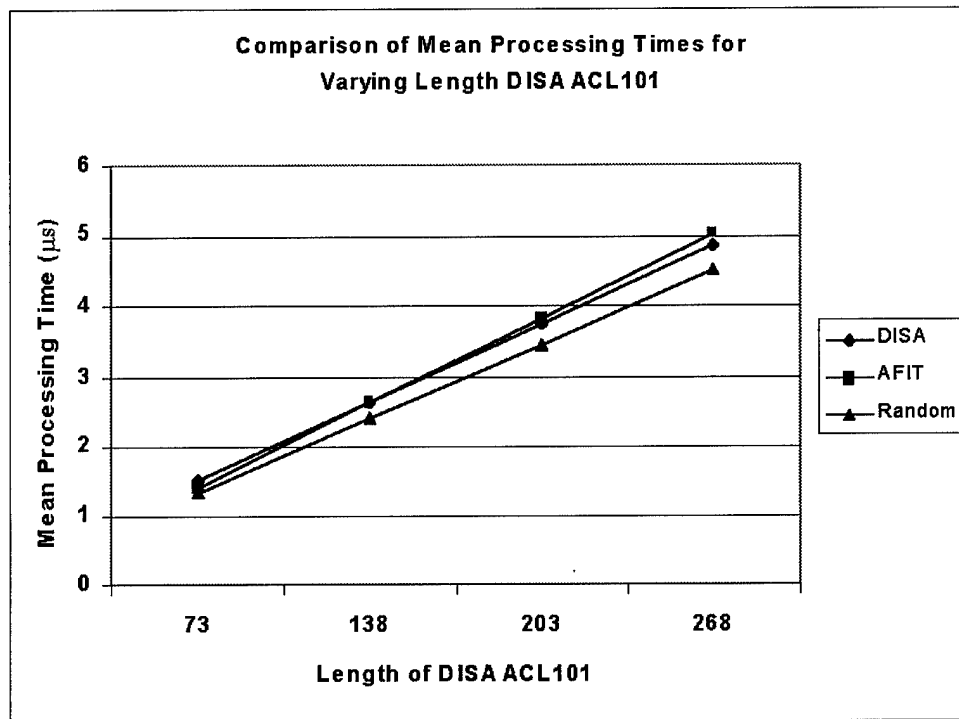
Figure 6. Comparison of Mean Processing Times for Varying Length AFIT ACL101

Figure 7 merges the results shown in Figure 6 into a single line representing the overall mean packet processing times associated with varying lengths of the AFIT access control list. As pointed out earlier, the results show a linear growth in the mean processing time.

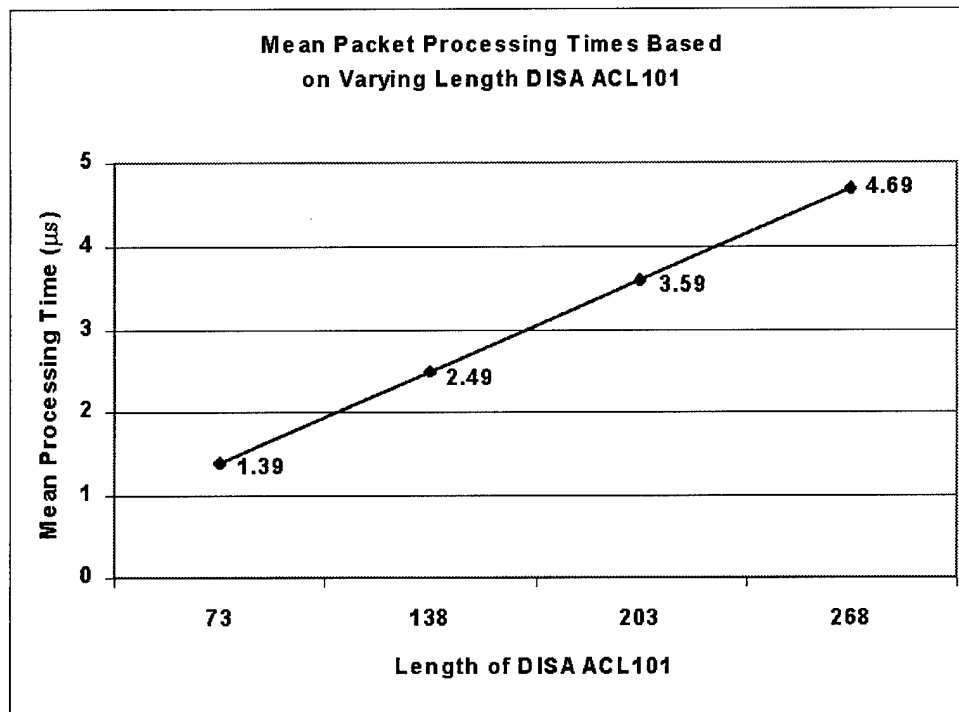


**Figure 7. Mean Processing Times for Varying Length AFIT ACL101**

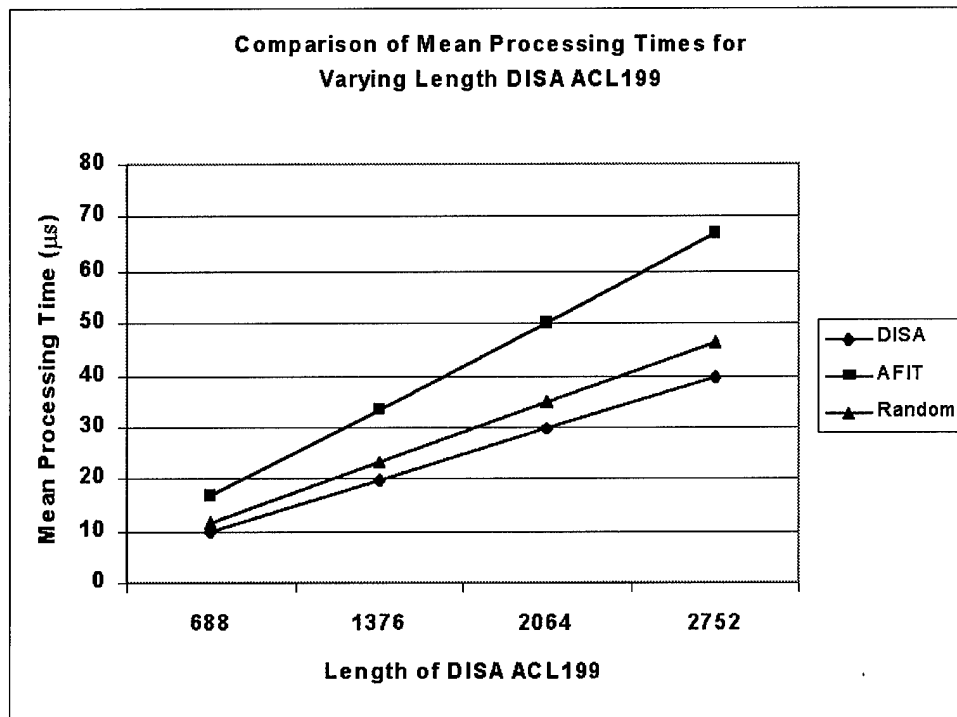
The next four charts were compiled in the same manner as described for Figures 6 and 7; they depict the values computed for the two DISA ACLs (101 and 199). The following results identify a linear growth in the packet processing times.



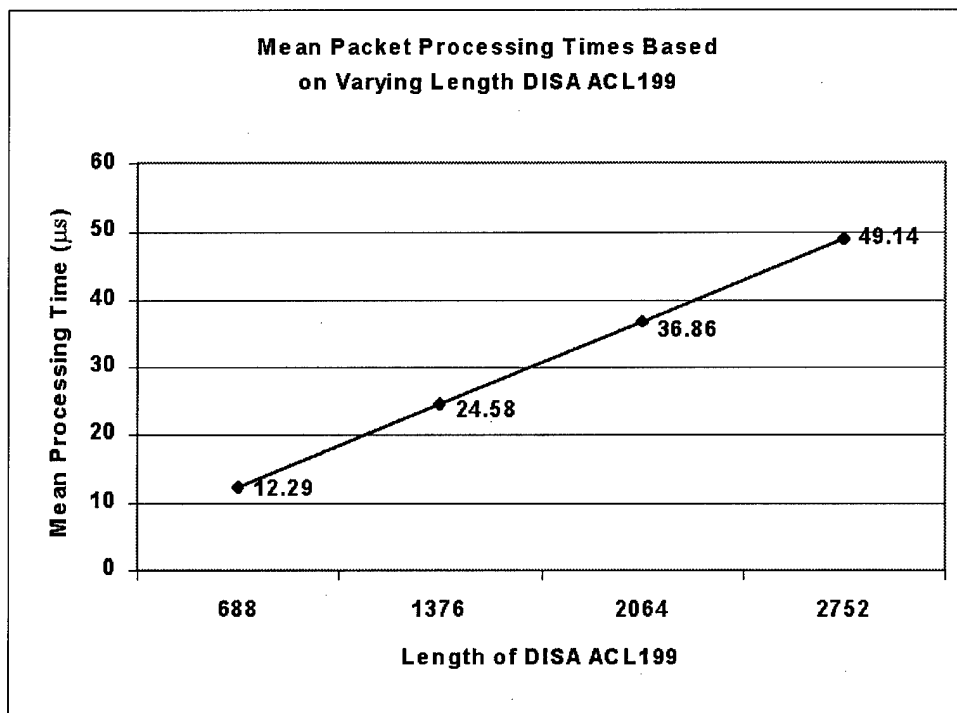
**Figure 8. Comparison of Mean Processing Times for Varying Length DISA ACL101**



**Figure 9. Mean Processing Times for Varying Length DISA ACL101**



**Figure 10. Comparison of Mean Processing Times for Varying Length DISA ACL199**



**Figure 11. Mean Processing Times for Varying Length DISA ACL199**



### 5.2.2 Implications of Findings

To put the results shown above into perspective, it is necessary to reiterate that the ACL Model represents a single component of a router. Therefore, the packet latency determined by the ACL Model is only a single component in the total service time per packet passing through the router. As mentioned in Section 4.2, this research focused on the routers located at the outer edge of a network. Therefore, the router is a gateway to the network and it is reasonable to expect the flow of packets to be large.

As an ACL grows, the packet latency increases, thus affecting the overall service time. The consequence of the packet service time increasing is the fact that incoming packets are required to wait in a queuing mechanism longer. The loss of packets could be the result of an access control list becoming too large. The queuing mechanism may become saturated and no longer capable of storing incoming packets due to the increased service times.

It is necessary for an organization to identify "choke points" within its router and catalog the various approaches to correct the problems. An organization may have to reduce the length of its ACL, or increase the size of the queuing mechanism on the router to compensate for the effects of surges in packet arrivals.

### 5.3 ACL Optimization Efforts

The study of the effects rearranging an access control list has on the overall processing time was very informative and beneficial. Within this section, the results of the optimization efforts on the AFIT and DISA ACLs are described. To aid in the description of the findings, tables are utilized to identify changes in processing times

based on the optimization efforts. A complete set of supplemental graphs designed to support the tables and text can be found in Appendix D.

The following data quantifies the output of the ACL Model. The results of this research are limited to the sample data and access control lists provided for this thesis effort.

### 5.3.1 Simulation of the AFIT ACL and the Results

#### 5.3.1.1 *Characterization of the Original AFIT ACL*

The AFIT ACL in its original form uses a combination of a “permit-all/deny all” policy. The ACL denies packets with specific IP addresses while implicitly permitting the rest. A TCP or UDP packet is denied if it is destined for a certain machine; however, the packet is allowed into the AFIT Network if the destination port meets specific criteria. All other TCP and UDP packets are implicitly denied. All ICMP packets are denied. The ACL is designed to completely isolate sub-networks within the AFIT Network. The ACL in general is organized to a certain extent and optimized for ease of maintenance; however, there is room for improvement in terms of ordering and the effective use of terminator entries.

Table 11 identifies the mean packet latency achieved by running the ACL Model with the two input files and three randomly generated data sets identified in Table 10, against the original AFIT access control list. The values in the “Overall Mean” column are considered baseline values and are used to compare the results of the various optimization attempts. Refer to Figures 27 – 31 in Appendix D for a graphical representation of the protocols grouped by processing time.

**Table 11. Mean Processing Time per Packet ( $\mu$ s) – AFIT ACL Original**

| Input Data    | Protocol |        |        |        | Overall Mean |
|---------------|----------|--------|--------|--------|--------------|
|               | IP       | ICMP   | TCP    | UDP    |              |
| AFIT          | 1.2974   | 1.0512 | 2.2107 | 2.4048 | 1.3507       |
| Random Seed 1 | 1.2463   | 1.0248 | 1.9183 | 1.9376 | 1.5253       |
| Random Seed 2 | 1.2465   | 1.0248 | 1.9187 | 1.9373 | 1.5271       |
| Random Seed 3 | 1.2465   | 1.0248 | 1.9187 | 1.9377 | 1.5350       |
| DISA          | N/A      | 1.0248 | 1.9189 | 1.9368 | 1.9092       |

### 5.3.1.2 Data Sample Frequency Analysis

A frequency analysis of the incoming AFIT IP packet headers provided the necessary data to aid in the optimization attempt of the AFIT access control list. The first type of frequency analysis conducted consisted of identifying source addresses specifically singled out in the ACL. The AFIT IP packet headers available had no matching entries in the ACL, since the AFIT ACL filtered out the ones that did match. Therefore, the next step involved accomplishing a frequency analysis based on the distribution of packet headers representing each type of protocol in the input files and randomly generated data. As mentioned earlier, two different input files and three randomly generated sets of IP packet headers were used throughout this thesis effort, each consisting of 11,443 entries.

#### 5.3.1.2.1 AFIT ACL Optimization Attempt and Results

As shown in Table 10, the majority of the incoming packets to AFIT were of the type IP, followed by the TCP, UDP and ICMP protocols. Therefore, the access control list optimization attempt involved moving all ACL entries responsible for singling out IP packet headers to the top of the list. The IP grouping was terminated with the following terminator:

```
access list 101 permit ip 0.0.0.0 255.255.255.255 0.0.0.0 255.255.255.255
```

The above terminator was originally located at the end of the ACL and was designed to implement the “permit all IP” policy. In a similar manner the TCP and UDP entries were grouped together and placed in their proper position within the ACL based on the packet distribution. At the end of the TCP and UDP groups a terminator entry was entered, designed to deny packets not meeting the defined exit criterion. The purpose of utilizing terminator entries was to prevent an incoming packet from being scanned beyond the applicable protocol section in the ACL.

The results of the optimization attempt of the AFIT ACL were very positive, the processing time (packet latency) associated with each packet dropped in all cases except UDP packets. The IP packet mean processing time fell from 1.2974  $\mu$ s to 0.7095  $\mu$ s, a decrease in IP packet latency of 45%. Table 12 identifies the mean packet latency achieved by running the two input files and three randomly generated data sets against the “optimized” AFIT ACL101. The overall mean processing time for AFIT packets dropped by more than 0.56  $\mu$ s or 41%. Refer to Figures 32 – 36 in Appendix D for a graphical representation of the protocols grouped by processing time.

**Table 12. Mean Processing Time per Packet ( $\mu$ s) - AFIT ACL Optimization Trial**

| Input Data    | Protocol |        |        |        | Overall Mean |
|---------------|----------|--------|--------|--------|--------------|
|               | IP       | ICMP   | TCP    | UDP    |              |
| AFIT          | 0.7095   | 0.8064 | 1.9803 | 2.4624 | 0.7879       |
| Random Seed 1 | 0.7207   | 1.0824 | 2.0238 | 2.0433 | 1.4571       |
| Random Seed 2 | 0.7210   | 1.0824 | 2.0243 | 2.0429 | 1.4575       |
| Random Seed 3 | 0.7209   | 1.0824 | 2.0243 | 2.0433 | 1.4727       |
| DISA          | N/A      | 1.0824 | 2.0245 | 2.0424 | 2.0140       |

The decreases in the overall mean packet latency for the randomly generated data sets were to be expected. Although the organization of the AFIT ACL was tuned for the arrival of AFIT packet headers, the use of terminator entries made it possible to achieve

performance gains with the randomly generated packet headers. Table 10 contains the protocol distribution for the randomly generated packets. The overall mean latency increased for packets destined for DISA routers. The AFIT ACL was organized based on a large number of IP packets arriving at the router interface; the DISA input data set contained zero (0) IP packets and the majority of its contents were of the type TCP. Therefore, all of the DISA TCP packets were required to be scanned against the entries designed to capture IP packets (protocol check) in addition to passing through all of the TCP checks. Outside of grouping the ACL entries by protocol and placing them in the appropriate order, the primary optimization technique responsible for the decrease in overall packet latency was attributed to the use of terminator entries.

#### 5.3.1.2.2 AFIT ACL Modification and Results

This test involved moving the single ICMP entry to the top of the ACL, while preserving the order of the access control list as proposed in the preceding section. This experiment was designed to illustrate the impact a single misplaced ACL entry can have on the overall mean processing time.

The results of moving the ICMP entry to the top of the list increased the overall packet latency in all cases except UDP packets. Table 13 identifies the mean packet latency achieved by running the two input files and three randomly generated data sets against the modified form of AFIT ACL. The overall mean rose 0.0128  $\mu$ s or 1% as compared to the results of the optimization attempt. This increase in packet latency represents a small increase in the overall mean calculated in the previous test. Refer to Figures 37 – 41 in Appendix D for a graphical representation of the protocols grouped by processing time.

**Table 13. Mean Processing Time per Packet ( $\mu$ s) – AFIT ACL Modification Trial**

| Input Data    | Protocol |        |        |        | Overall Mean |
|---------------|----------|--------|--------|--------|--------------|
|               | IP       | ICMP   | TCP    | UDP    |              |
| AFIT          | 0.7250   | 0.1008 | 1.9803 | 2.4480 | 0.8007       |
| Random Seed 1 | 1.3375   | 1.0680 | 2.0094 | 2.0289 | 1.6044       |
| Random Seed 2 | 1.3378   | 1.0680 | 2.0099 | 2.0285 | 1.6063       |
| Random Seed 3 | 1.3981   | 0.9792 | 1.5008 | 1.4814 | 1.3409       |
| DISA          | N/A      | 1.0680 | 2.0101 | 2.0280 | 1.9996       |

### 5.3.1.3 Summary of the AFIT Results and Recommendations

The majority of the processing time improvements were derived by simply reordering the original ACL based on a frequency analysis of the incoming packet header's protocol type and the use of terminator entries. The appropriate terminator statement, dependent upon an organization's security policy, was placed at the end of each protocol section. If the appropriate protocol group does not capture the packet, the use of a terminator entry prevents the packet from traversing the remaining ACL entries. Table 14 represents the total processing times for each input data to pass through the respective ACL, all times are in milliseconds. The total processing time of the AFIT input data dropped from 15.45 ms for the original ACL to 9.01 ms for the "optimized" ACL, thus representing a performance gain of 60%.

**Table 14. Total Processing Times for Input Data by Various ACLs (ms)**

| Access Control List        | Input Data |               |               |               |         |
|----------------------------|------------|---------------|---------------|---------------|---------|
|                            | AFIT       | Random Seed 1 | Random Seed 2 | Random Seed 3 | DISA    |
| AFIT – Original Form       | 15.4563    | 17.4542       | 17.4743       | 17.5651       | 21.8465 |
| AFIT – Optimization Effort | 9.0160     | 16.6738       | 16.6785       | 16.8528       | 23.0466 |
| AFIT – Modification        | 9.1625     | 18.3590       | 18.3815       | 15.3439       | 22.8818 |

Based upon the frequency analysis of the packet headers provided by AFIT and the testing of various ACL optimization attempts, the following set of recommendations are offered:

- Consolidate all of the IP entries into a single grouping and move the entries to the top of the access control list. Based on the security climate of AFIT, place a “permit” all others entry at the end of the IP grouping. Such as:

```
access-list 101 permit ip 0.0.0.0 255.255.255.255 0.0.0.0 255.255.255.255
```

- Consolidate all of the TCP entries into a single grouping and place them immediately after the IP group. Once again, ensure a terminating statement is placed at the end of the TCP section to capture TCP packets not already singled out by the previous entries.
- Entries related to the control of UDP packets should be handled in the same manner as the TCP entries and placed immediately following the TCP group.
- The final two entries within the AFIT ACL focuses on capturing ICMP packets; the first entry designed to permit ICMP packets destined for the AFIT Network, while the second entry denies all other ICMP packets.
- The second ICMP entry is not needed since access control lists implemented in a Cisco router implicitly “deny all” packets passing through the entire list.  
  
However, by explicitly stating the desired outcome of all other ICMP packets, the entry adds clarity as to how packets are ultimately handled by the access control list.

### 5.3.2 Simulation of the DISA ACL and the Results

#### 5.3.2.1 Characterization of the Original DISA ACL

The DISA ACL in its original form uses a combination of “permit” and “deny” policies. For example, a set of IP entries is designed to deny specific IP addresses, while permitting access to all others. On the other hand, only TCP and UDP packets from certain addresses are permitted, while all other packets are denied. The DISA ACL permits all ICMP packets. In general, the entries of the DISA ACL are assembled indiscriminately. In the ACL there are several IP entries at the top and bottom, broken up with TCP and UDP entries in the middle. The TCP and UDP packets appear to be organized with the intent of easing ACL maintenance. Based on the results of the AFIT optimization efforts, the same types of recommendations apply for the DISA ACL.

Table 15 identifies the mean packet latency achieved by running the two input files and three randomly generated data sets, identified in Table 10, against the original DISA access control list. The values in the “Overall Mean” column serve as baseline values and are used to compare the results of the various optimization attempts. Refer to Figures 42 – 46 in Appendix D for a graphical representation of the protocols grouped by processing time.

**Table 15. Mean Processing Time per Packet ( $\mu$ s) – DISA ACL Original**

| Input Data    | Protocol |        |        |        | Overall Mean |
|---------------|----------|--------|--------|--------|--------------|
|               | IP       | ICMP   | TCP    | UDP    |              |
| DISA          | N/A      | 0.9792 | 1.5119 | 1.4899 | 1.4992       |
| Random Seed 1 | 1.3984   | 0.9792 | 1.5009 | 1.4815 | 1.3381       |
| Random Seed 2 | 1.3985   | 0.9792 | 1.5005 | 1.4815 | 1.3397       |
| Random Seed 3 | 1.3981   | 0.9792 | 1.5008 | 1.4814 | 1.3409       |
| AFIT          | 1.4118   | 0.9792 | 1.5002 | 1.4842 | 1.4155       |



### *5.3.2.2 Data Sample Frequency Analysis*

A frequency analysis of the incoming DISA packet headers provided the necessary data to aid in the optimization of the DISA access control list. The first type of frequency analysis conducted consisted of identifying source addresses specifically singled out in the ACL. The DISA packet headers available had no matching entries in the ACL. Therefore, the next step involved accomplishing a frequency analysis based on the distribution of packet headers representing each type of protocol in the input file. Table 10 categorizes the contents of the DISA input file, based on the packet header's protocol type.

#### *5.3.2.2.1 DISA ACL Optimization Attempt and Results*

As shown in Table 10, the majority of the incoming packets to DISA were of the type TCP, followed by the UDP and ICMP protocols. There were no IP packets in the data set from DISA. Therefore, the ACL optimization attempt for the DISA ACL involved moving entries responsible for singling out TCP packet headers to the top of the list. Similarly, the UDP, ICMP and IP entries were grouped together and placed in their proper position within the ACL based on the packet distribution. At the end of each of these groups a terminator entry was placed, to prevent an incoming packet from being scanned beyond the applicable protocol section in the access control list.

The results of the optimization attempt were very positive; latency associated with each packet dropped in all cases. The mean processing time of TCP packets fell from 1.51  $\mu$ s to 0.88  $\mu$ s, a decrease in TCP packet latency of 42%. Table 16 identifies the mean packet latency achieved by running the two input files and three randomly generated data sets against the "optimized" form of the DISA ACL. The overall mean for

DISA packets dropped by more than 0.55  $\mu$ s or 36%. Refer to Figures 47 – 51 in Appendix D for a graphical representation of the protocols grouped by processing time.

**Table 16. Mean Processing Time per Packet ( $\mu$ s) – DISA ACL Optimization Trial**

| Input Data    | Protocol |        |        |        | Overall Mean |
|---------------|----------|--------|--------|--------|--------------|
|               | IP       | ICMP   | TCP    | UDP    |              |
| DISA          | N/A      | 0.7632 | 0.8856 | 1.1805 | 0.9459       |
| Random Seed 1 | 1.4272   | 0.7632 | 1.2530 | 1.3870 | 1.2072       |
| Random Seed 2 | 1.4273   | 0.7632 | 1.2447 | 1.3859 | 1.2069       |
| Random Seed 3 | 1.4271   | 0.7632 | 1.2325 | 1.3822 | 1.2010       |
| AFIT          | 1.4406   | 0.7632 | 0.8738 | 1.1746 | 1.4122       |

The decreases in the overall mean packet latency for the randomly generated data sets were to be expected. Although the organization of the DISA ACL was tuned for the arrival of DISA packet headers, the use of terminator entries made it possible to achieve performance gains with the randomly generated packet headers. Table 10 contains the protocol distribution for the randomly generated packets. The overall mean latency increased for packets destined for AFIT routers. The DISA ACL is organized based on a large number of TCP packets arriving at the router interface; the AFIT input data set contained few TCP packets while the majority of its contents were of the type IP.

Therefore, all of the AFIT IP packets were scanned against the entries configured to capture TCP, UDP and ICMP packets (protocol check) in addition to passing through all of the IP checks at the end of the ACL. Outside of grouping the ACL entries by protocol and placing them in the appropriate order, the primary optimization technique responsible for the decrease in overall packet latency can be attributed to the use of terminator entries.

#### 5.3.2.2.2 DISA ACL Modification and Results

This test involved moving the single ICMP entry from its position between the UDP and IP entries to the top of the ACL, while preserving the order of the access control list

as proposed in the preceding section. This experiment was designed to illustrate the impact a single misplaced ACL entry can have on the overall mean processing time.

Table 17 identifies the mean packet latency achieved by utilizing the two input files and three randomly generated data sets when scanned against the modified DISA ACL. The impact of this modification is very small, the overall mean rose 0.0041  $\mu$ s (0.4%) as compared to the results of the optimization effort. Refer to Figures 52 – 56 in Appendix D for a graphical representation of the protocols grouped by processing time.

**Table 17. Mean Processing Time per Packet ( $\mu$ s) – DISA ACL Modification Trial**

| Input Data    | Protocol |        |        |        | Overall Mean |
|---------------|----------|--------|--------|--------|--------------|
|               | IP       | ICMP   | TCP    | UDP    |              |
| DISA          | N/A      | 0.1008 | 0.9000 | 1.1949 | 0.9500       |
| Random Seed 1 | 1.4272   | 0.1008 | 1.2600 | 1.3940 | 1.0433       |
| Random Seed 2 | 1.4273   | 0.1008 | 1.2519 | 1.3930 | 1.0456       |
| Random Seed 3 | 1.4271   | 0.1008 | 1.2399 | 1.3894 | 1.0399       |
| AFIT          | 1.4406   | 0.1008 | 0.8882 | 1.1890 | 1.4115       |

### 5.3.2.3 Summary of the DISA Results and Recommendations

The conclusions concerning optimization techniques for the DISA access control list are similar in content as those provided in Section 5.3.1.3 for the AFIT ACL. By reordering the ACL entries based on a frequency analysis of the protocols associated with the incoming packets and the effective use of terminator entries, the overall packet latency can be minimized. Table 18 represents the total processing times for the input data to pass through the respective ACL, all times are in milliseconds. The total processing time of the DISA input data dropped from 17.56 ms for the original ACL to 10.82 ms for the “optimized” ACL, thus representing a performance gain of 63%.

**Table 18. Total Processing Times for Input Data by Various ACLs (ms)**

| Access Control List        | Input Data |               |               |               |         |
|----------------------------|------------|---------------|---------------|---------------|---------|
|                            | DISA       | Random Seed 1 | Random Seed 2 | Random Seed 3 | AFIT    |
| DISA – Original Form       | 17.1554    | 15.3117       | 15.3306       | 15.3439       | 16.1978 |
| DISA – Optimization Effort | 10.8231    | 13.8139       | 13.8101       | 13.7429       | 16.1596 |
| DISA – Modification        | 10.8708    | 11.9390       | 11.9652       | 11.8998       | 16.1522 |

Based upon the frequency analysis of the packet headers provided by DISA and the testing of various ACL optimizations, the following set of recommendations are offered:

- Consolidate all of the TCP entries into a single grouping and move the entries to the top of the access control list. Based on the security climate of DISA, place a “deny” all others entry at the end of the TCP grouping. Such as:

```
access-list 101 deny tcp 0.0.0.0 255.255.255.255 0.0.0.0 255.255.255.255 neq 0
```

- Consolidate all of the UDP entries into a single grouping and place them immediately after the TCP entries. Once again, ensure a terminating statement is placed at the end of the UDP section to capture UDP packets not already singled out by the previous entries.
- Move the single ICMP entry, permitting all incoming ICMP packets to pass through the router, immediately after the UDP section.
- The final group of entries in the DISA ACL is designed to control the flow of IP packets through the router. The last entry of the IP group (and consequently the ACL) is the “Permit” all other IP packets.

#### **5.4 ACL Construction Guidelines to Enhance Router Performance**

The following six steps identify a set of guidelines to follow when creating an Extended IP Access Control List or modifying an existing one, thereby ensuring router

performance is maximized. The final step highlights the importance of fine tuning the ACL based on local needs.

Step 1: Accomplish a frequency analysis.

The first step to take in optimizing an Extended IP Access Control List involves accomplishing a frequency analysis of the IP packets arriving at the router interface, where the ACL to be evaluated is assigned. The data collected should be from various times of the day and week, providing a well-rounded sampling of packets to be analyzed. During the frequency analysis, the individual responsible for analyzing the incoming IP packets accomplishes two steps, the first is to identify the distribution of packets based on their protocol type. The second step in analyzing the data entails identifying source addresses specifically singled out in the ACL and noting their arrival frequency.

Step 2: Group ACL entries by protocol.

This step involves modifying the ACL from its original form, by arranging all like protocol ACL entries into distinct groups. In most instances there will be one group of entries for each protocol covered by the Extended IP ACLs (IP, TCP, UDP and ICMP).

Step 3: Position groups in proper order within ACL based on frequency analysis data.

Based on the data acquired during the frequency analysis phase, take the various groups of ACL entries created in the last step and place them in the proper order in the ACL based on the protocol distribution.

To clarify, if the majority of the incoming packets to the router interface were of the type IP, followed by the UDP, TCP, and ICMP protocols, move the entire group of IP ACL entries to the top of the ACL. The UDP, TCP and ICMP groups would follow the IP entries in their respective order.

If it is determined from the analysis of the sample packets that a majority of the packets arrive from a single source then it may be necessary to deviate from the strict group setup. It would still be beneficial to keep the groups by protocol; however, place a single entry at the top of the ACL, to capture the specific source address and process it immediately. According to Phillip Harris, Senior Consulting Engineer for Cisco, if certain addresses frequently arrive against a specific ACL, performance can be increased if the applicable entry in the ACL is moved up in the list [Har98]. By moving the entry up in the access control list, the router matches the incoming packet to the entry sooner and less overall processing is required. This part of Step 3 was not characterized in our set of experiments, since there was no dominant set of packets identified as coming from a single address.

#### Step 4: Proper use of terminator entries.

When grouping ACL entries by protocol type, it is imperative that each group is terminated with the proper terminator entry. Effective use of the terminator entries can reduce overall packet processing times. By placing a terminator entry at the end of a protocol group, it prevents incoming packets from being scanned beyond the applicable protocol section. Deciding whether to “deny” or “permit” packets not specifically singled out by the preceding ACL entries should be based on an organization’s security policy. The use of terminators also makes an ACL easier to read. In addition, the extra entries make the final status of packets not singled out in the appropriate protocol group very explicit. In general, when working with “wildcard” masks in the terminator entries, care must be taken by the ACL administrator to avoid changing the filtering outcome.

#### Step 5: Focus on order of entries within each group.

Common sense dictates that the ordering of the entries within each protocol group may be important. The maintainer of an ACL should place entries directly supporting the interests of partners or clients ahead of entries designed to prevent access to known malefactors. It is more advantageous to make a hacker's incoming packets wait, versus a valid business transaction. During a Denial of Service attack, everyone has to wait; however, this methodology reduces service times during normal network operations.

Step 6: Fine-tune the ACL based on local needs.

The first five steps identified the general guidelines to follow when initially trying to optimize an access control list. This step is designed to bring to light other situations that can be taken into consideration when creating or modifying an ACL. System administrators can accomplish their own tests, and fine-tune the ACL based on local operations. Look at the organization's security policy and ensure the ACL meets the described objectives. Ask the crucial question, "Are the network customers' and users' needs being met by the current ACL configuration?"

## **5.5 Summary**

The access control list simulation system, ACL Model, was designed to test the effects growing ACLs have on network router performance. The data presented in the first part of this chapter showed a linear growth rate in the packet processing times. The linear growth in packet latency was not expected, since the purpose of the research was to identify the point where an ACL becomes too large and system performance is severely degraded. However, this research and data does show that as an access control list grows, the mean packet latency increases. An increase in packet latency caused by the ACL component of a router increases the overall service time required to process each packet

passing through the router. Increasing the service time can have a negative impact on operations if packets are lost due to the queuing mechanism becoming full and not accepting any other incoming packets.

The second portion of the chapter detailed how the ACL Model proved to be beneficial in terms of identifying guidelines designed to improve router performance. Section 5.3 focused on the results of various optimization efforts of the AFIT and DISA ACLs. It was determined that the most effective performance enhancing optimization was a combination of grouping the entries by protocol, ordering the groups in the ACL based on the protocol distribution and the use of terminator entries. Section 5.4 focused on identifying six steps an individual responsible for maintaining ACLs can follow to enhance router performance.



## **6. Conclusion**

### **6.1 Introduction**

The purpose of this research effort was to determine how the growth of an access control list affects packet flow and router CPU consumption, and identify the specific length of an access control list such that overall router performance is degraded. The secondary goal was to provide insights on how access control lists can be optimized.

This thesis described the ACL Model, a tool designed to simulate the packet validation component contained within a network router. There are many components contributing to the overall service time associated with a packet passing through a network router; however, the ACL Model only computed the resultant latency of a packet passing through the packet validation component of a router.

This chapter highlights the final results achieved during this research effort. It also addresses a possible future research topic. The chapter concludes with summary remarks.

### **6.2 Conclusions**

This research was designed to accomplish the goals described above. Conclusions regarding each goal are presented individually, starting with findings on the performance issues. This section concludes with a discussion of findings concerning optimization techniques.

#### **6.2.1 Summary of Performance Issues**

A comparison of the applicable simulations accomplished for this research effort showed that the packet latency grew in a linear fashion as the length of the ACL grew. It was expected that the packet processing times increased as the assigned ACL grew, linear

growth in packet latency was not expected (refer to Figure 4 in Chapter 3). Exponential growth was anticipated in order to identify a discernable “knee” in the performance curve. If a “knee” could be determined, then identification of the length of an ACL where router CPU consumption became noticeably degraded could be made. Since the growth of the packet latency was linear, it was impossible to identify where the network router CPU had to work harder based on the length of the ACL. The linear growth seen in the various experiments are considered worst case scenarios. By utilizing the optimization guidelines presented in Chapter 5, system administrators can fine-tune their ACLs and expect sublinear growth rates in the mean packet latency.

In general, as an ACL grows the packet latency increases, thus affecting the overall service time. As the overall service times increase, incoming packets are delayed in a front-end queuing mechanism. The end result of packets being held in a queue could be the loss of packets, due in part by the queuing mechanism becoming saturated and no longer capable of storing incoming packets. System administrators must be ever vigilant in ensuring their ACLs are not affecting the performance of their router.

### 6.2.2 Summary of the Optimization Efforts

The next several paragraphs highlight research findings concerning the methodology to follow in optimizing an ACL. The study of the effects rearranging an access control list has on the overall processing time was very informative and beneficial.

In both cases where optimization of the AFIT and DISA ACLs occurred, the overall mean packet latency was reduced by more than half a microsecond. A decrease of  $0.5\ \mu\text{s}$  doesn't appear to be a large value; however, if an average packet requires approximately  $1.49\ \mu\text{s}$  to pass through the packet validation component of a router the decrease appears

to be significant. The previous figures represent a processor improvement of approximately 34%. The total processing times for the same cases characterize an overall performance gain of approximately 61%.

The reduction in processing time was realized by accomplishing several steps. The first step in optimizing an ACL involved accomplishing a frequency analysis of the incoming packets. Upon completion of the frequency analysis, all of the entries in the ACL were grouped by protocol type; each protocol grouping was moved to the appropriate location in the ACL as identified by the data analysis. The final step involved utilizing terminator entries after each grouping to trap packets not meeting the criteria specified by the preceding ACL entries.

### **6.3 Possible Model Inadequacies**

As explained in Chapter 3, the simulation route was chosen over both analytical modeling and the measurement techniques. It may be beneficial to locate a router and run similar experiments as accomplished in this thesis effort; it would be interesting to compare the results of the two different techniques. Since the details of the operations occurring within a router are proprietary, it was necessary to formulate the most logical flow of packets through a router. Even though the ACL Model was validated by an individual from Cisco, the proprietary nature of the internal components made it difficult to ascertain the instruction counts; the instruction counts calculated for each module could be a too high or too low. In an attempt to demonstrate the robustness of the conclusions, a sensitivity analysis was accomplished. The analysis showed the processing time would rise as the instruction count increased, as expected and more importantly, the

mean processing times always grew in a linear fashion regardless of the specific mix of instruction counts.

#### **6.4 Areas of Model Improvement**

The ACL Model did not implement any ACL processing enhancements since these enhancements are not utilized at AFIT or DISA. However, further research in the area of ACL performance may warrant the implementation of the enhancements discussed in Section 2.5. The process enhancements include the *Fast Switching Path* for standard IP ACLs, *NetFlow Switching*, *Distributed Switching*, and *Cisco Express Forwarding* for Extended IP Access Control Lists.

#### **6.5 Recommendation for Future Research**

One of the biggest concerns over the next couple of years is the introduction of Internet Protocol version 6 (IPv6). IPv4 is currently in use around the world utilizing 32-bit addressing. The move to IPv6 increases the addressing scheme to 128-bits. Of interest is the effect 128-bit addressing has on access control lists and the amount of time required to process incoming packets.

In an attempt to ascertain the affect 128-bit addressing would have on the mean processing time per packet validation, the ACL Model was used. Using the ACL Model to determine the packet latency in this case is not very accurate without modifying the model to handle the wider addresses; however, it provides a rough figure to build upon. To accomplish this test, the *Mask Check* instruction count was increased to sixteen versus four. The sixteen instructions would be equivalent to accomplishing four separate mask checks. The results of the simulation showed the overall mean processing time again

increases linearly. To properly address the 128-bit addressing topic, modifications to the ACL Model are required.

## **6.6 Summary Remarks**

As the world becomes more interconnected through the use of the Internet, it is imperative organizations take the proper steps to ensure the security of their networks is maintained. Organizations can no longer isolate their networks from the rest of the world and still remain competitive. An organization willing to compete in the world market must take the necessary precautions to protect its networks, the systems located on those networks, and its mission critical data. There are performance issues associated with the use of access control lists (ACL); however, if ACLs are implemented properly and periodically reviewed, a secure network can be attained.

## ***Appendix A. Details of IP Access Control Lists***

This appendix is designed to aid the reader in identifying the various parts of an access control list. Not all of the intricacies are identified; however, the more commonly used notations are characterized. The sample entries used in this appendix were compiled from various sources to ensure the many differing constructs are depicted. Several examples of standard IP access control lists are given. However, contingent upon the primary focus of this thesis effort, the majority of the examples contained here are based on Extended IP Access Control Lists.

### **A.1 IP Access Control Lists**

Standard IP Access Control List specifications:

|  |
|--|
| <code>access-list 1-99 {permit deny} {address} {mask}</code> |
|--|

In the following example, network 109.90.0.0 is a Class A network whose second octet specifies a subnet. The third and fourth octets of a network 109.90.0.0 address specify a particular host. Using access list 2, the router would accept one address on host 20 and reject all others on that host. The last line of the list depicts the router accepting addresses on all other network 109.90.0.0 subnets.

```
access-list 2 permit 109.90.20.3
access-list 2 deny 109.90.20.0 0.0.0.255
access-list 2 permit 109.90.0.0 0.0.255.255
```

Standard IP access control list entries contain an implicit mask. For instance, if the mask is omitted from an associated IP host address access list specification, 0.0.0.0 is assumed to be the mask. Consider the following example configuration:

```
access-list 1 permit 0.0.0.0
access-list 1 permit 109.90.0.0
```

```
access-list 1 deny 0.0.0.0 255.255.255.255
```

For this example, the following masks are implied in the first two lines:

```
access-list 1 permit 0.0.0.0 0.0.0.0
```

```
access-list 1 permit 109.90.0.0 0.0.0.0
```

The last line in the configuration (using the deny keyword) can be left off, since IP ACLs implicitly deny all other access. This is equivalent to ending the ACL with the following command statement:

```
access-list 1 deny 0.0.0.0 255.255.255.255
```

## A.2 Extended IP Access Control List

Extended IP Access Control List specifications:

|   |
|---|
| <pre>access-list 100-199 {permit deny} {ip tcp udp icmp} {source} {source-mask} {dest} {dest-mask} {lt gt eq neq dest-port}</pre> |
|---|

In the following example, the first line permits any incoming TCP connections to hosts 109.90.0.0 with destination port addresses greater than 1023. The second line permits incoming TCP connections to the SMTP port (25) of host 109.90.1.2. The last line permits incoming ICMP messages for error feedback.

```
access-list 102 permit tcp 0.0.0.0 255.255.255.255 109.90.0.0 0.0.255.255 gt 1023
access-list 102 permit tcp 0.0.0.0 255.255.255.255 109.90.1.2 0.0.0.0 eq 25
access-list 102 permit icmp 0.0.0.0 255.255.255.255 109.90.0.0 255.255.255.255
```

Similar to the standard ACLs, the last line in the configuration (using the deny keyword) can be left off, since IP access control lists implicitly deny all other access.

This is equivalent to completing the ACL with the following entry:

```
access-list 102 deny ip 0.0.0.0 255.255.255.255 0.0.0.0 255.255.255.255
```

One final example of an Extended IP Access Control List is provided below. There are minor differences in the syntax used, to show variations that can be utilized to ease ACL maintenance. To illustrate this idea, note that the first line of the example is in the

same format as used in the previous examples. However, for the TCP and UDP entries, 0.0.0.0 255.255.255.255 has been replaced with the keyword **any**. In essence the TCP entry is stating, permit packets from anywhere heading for destination of 109.90.20.13 as long as they arrive on port address 80. The final entry denies ICMP packets from anywhere and destined for any destination to pass through the router.

```
access-list 101 deny ip 0.0.0.0 255.255.255.255 109.90.20.13 0.0.0.0
access-list 101 permit tcp any 109.90.20.13 0.0.0.0 eq 80
access-list 101 permit udp any 109.90.20.0 0.0.0.255 eq 80
access-list 101 deny icmp any any
```

### A.3 Summary of Numerical Ranges

In the few examples presented above, the numbers after the phrase “access-list” ranged from 1 to 102. In Table 19, the various protocols are identified, with each one having a designated range of valid numbers. When reading an ACL, a person may initially look at the number associated with the entries and determine what type of protocol the ACL is designed to support.

**Table 19. Summary of Numerical Ranges**

| Protocol           | Range      |
|--------------------|------------|
| IP                 | 1--99      |
| Extended IP        | 100--199   |
| Ethernet type code | 200--299   |
| DECnet             | 300--399   |
| XNS                | 400--499   |
| Extended XNS       | 500--599   |
| AppleTalk          | 600--699   |
| Ethernet address   | 700--799   |
| Novell             | 800--899   |
| Extended Novell    | 900--999   |
| Novell SAP         | 1000--1099 |



## Appendix B. Samples of Input ACLs and Packet Header Data

This appendix is designed to provide the format of the input data files. The data provided below shows how the raw data was transformed into data useable by Designer. The *Single ACL Entry* and *Packet Header* data structures were defined in Chapter 4, refer to Table 8 and Table 9 as necessary to identify each field within the final format.

### B.1 Single ACL Entry Data

Original format of an Extended IP ACL entry, designed to deny IP packets to a specific address:

```
access-list 101 deny ip 0.0.0.0 255.255.255.255 109.90.20.13 0.0.0.0
```

Final format:

```
0 0 0 0 0 0 255 255 255 255 109 90 20 13 0 0 0 0 0 0
```

Original format of an Extended IP ACL entry, designed to permit any TCP packet into a specific address and port:

```
access-list 102 permit tcp 0.0.0.0 255.255.255.255 128.88.1.2 0.0.0.0 eq 25
```

Final format:

```
1 2 0 0 0 0 255 255 255 255 128 88 1 2 0 0 0 0 3 25
```

### B.2 IP Packet Header Data

Original format of a sample IP packet header:

| SrcIf | SrcIPaddress  | DstIf | DstIPaddress   | Pr | SrcP | DstP | Pkts | B/Pk | Active |
|-------|---------------|-------|----------------|----|------|------|------|------|--------|
| Hs5/0 | 207.46.142.23 | Se2/7 | 131.48.192.183 | 06 | 0050 | 04FE | 5    | 41   | 0.4    |

Final Format:

```
2 207 46 142 23 131 48 192 183 80 1278
```

### **Appendix C. Access Control List Model**

To support this thesis, the Access Control List Model was built to simulate the packet validation process occurring within a network router. In this appendix, each Designer module of the ACL Model is discussed in detail to include an explanation of its function, a description of the data flow, and if applicable, a description of how the instruction count was calculated. The following table is a roadmap for this appendix; the model operation as explained in this appendix is top-down and meant to provide an overall view of the path an IP packet traverses during the packet validation process.

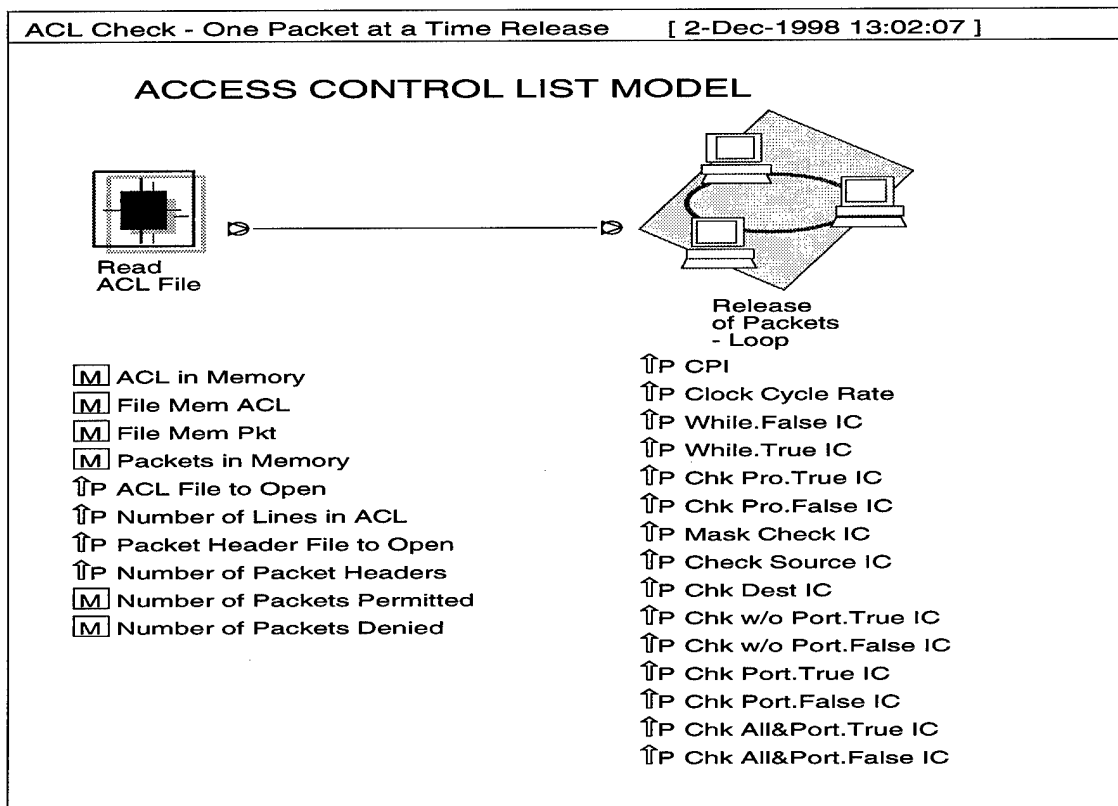
**Table 20. Summary of Modules by Operation**

| <b>Operation</b>                  | <b>Module Name</b>   |
|-----------------------------------|--|
| Initialize the Simulation         | Read ACL File  |
| Arrival of IP Packets into Router | Release of Packets – Loop<br>Read Packets – File<br>Read Packets – Random  |
| Packet Validation                 | While Loop<br>Single ACL from Vector<br>Check All Module<br>Check All & Port Module<br>Check Protocol<br>Check Source<br>Mask Check<br>Check Destination<br>Check Port<br>Check w/o Port |

A brief addendum concerning the terminology used throughout this appendix: the terms “module” and “block” have two distinct meanings. *Module* will be used to refer to a grouping of Designer functions, performing several operations on a data structure traversing that component. A *block* is a primitive operation on a data structure, such as a delay function.

## C.1 Access Control List (ACL) Model

Due to inheritance, many modules use the same parameters as other modules. Since the same parameters may appear in several modules, they will each be described once in Table 21. The instruction count variables are discussed in detail with their respective modules. Finally, there is an in-depth discussion of a subset of model parameters and variables in the “Overview of the ACL Model Parameters” section of Chapter 4. Refer to it as necessary.



**Figure 12. ACL Model Top Level System Module**

The *ACL Check – One Packet at a Time Release* (Figure 12) is the ACL Model top-level system module. This module ties together the two components responsible for initializing the simulation at start time (Figure 13. Read ACL File Module) and the reading of packets from an external file to be forwarded to the packet validation software

(Figure 14. Release of Packets - Loop Module). In the table below, each variable present in the top-level schematic is identified and a summary of each provided.

**Table 21. Top Level ACL Model Variables**

| Simulation Variables        | Initial Value      | Purpose  |
|-----------------------------|--------------------|--|
| ACL in Memory               | Varying            | <i>ACL in Memory</i> is a vector of varying length (between different simulations) depending on the size of the ACL input file. Length of vector is calculated in the following manner: (Number of Lines in ACL File * 20). There are 20 elements in each ACL entry. |
| File Mem ACL                | Pointer            | <i>File Mem ACL</i> is a pointer used to maintain the proper position in the input ACL file, while reading values from it and placing them into the ACL vector.  |
| File Mem Pkt                | Pointer            | <i>File Mem Pkt</i> is a pointer used to maintain the proper position in the input IP data file. This pointer is extremely important since the entire file is not read all at once, only eleven values are read at a time from the file.                             |
| Packets in Memory           | Varying            | <i>Packets in Memory</i> is a vector of varying length (between different simulations) depending on the size of the packet header data file.   |
| Number of Packets Permitted | 0                  | <i>Number of Packets Permitted</i> is discussed in Section 4.3.3   |
| Number of Packets Denied    | 0                  | <i>Number of Packets Denied</i> is discussed in Section 4.3.3  |
| ACL File to Open            | Set by user        | <i>ACL File to Open</i> contains the directory path and name of the file to open for the simulation. The ACL input file is text based and the format is very specific. For details on the specific format of the file refer to Appendix B.                           |
| Number of Lines in ACL      | Set by user        | <i>Number of Lines in ACL</i> variable is discussed in Section 4.3.2.2.  |
| Packet Header File to Open  | Set by user        | <i>Packet Header File to Open</i> contains the directory path and name of the file to open for the simulation. The IP packet header input file is text based and the format is very specific. For details on the format of the file refer to Appendix B.             |
| Number of Packet Headers    | Set by user        | <i>Number of Packet Headers</i> variable is discussed in Section 4.3.2.2.  |
| CPI                         | 1.2                | <i>CPI</i> holds the clock cycles per instruction. This value is based on research [EsR91] and the particular RISC chip utilized in the Cisco 7500 series router. Additional information concerning this variable can be found in Section 3.4.2.                     |
| Clock Cycle Rate            | $5 \times 10^{-9}$ | <i>Clock Cycle Rate</i> identifies the speed of the microprocessor, clocked in nanoseconds. Documentation shows that the Cisco 7500 Series routers operate at 200 MHz. The equation for this conversion can be found in Section 3.4.2.                               |
| While.False IC              | 5                  | Instruction count for the While Loop module when the false path is followed.   |

| Simulation Variables  | Initial Value | Purpose   |
|-----------------------|---------------|---|
| While.True IC         | 4             | Instruction count for the While Loop module when the true path is followed.   |
| Chk Pro.True IC       | 1             | Instruction count for the Check Protocol module when the true path is followed.   |
| Chk Pro.False IC      | 2             | Instruction count for the Check Protocol module when the false path is followed.  |
| Mask Check IC         | 4             | Instruction count for the Mask Check module.  |
| Check Source IC       | 1             | Instruction count for the Check Source module.  |
| Chk Dest IC           | 1             | Instruction count for the Check Destination module.   |
| Chk w/o Port.True IC  | 3             | Instruction count for the Check w/o Port module when the true path is followed.   |
| Chk w/ Port.False IC  | 2             | Instruction count for the Check w/o Port module when the false path is followed.  |
| Chk Port.True IC      | 1             | Instruction count for the Check Port module if the true path is followed.   |
| Chk Port.False IC     | 2             | Instruction count for the Check Port module when the false path is followed.  |
| Chk All&Port.True IC  | 3             | Instruction count for the Check All & Port module when the true path is followed.   |
| Chk All&Port.False IC | 2             | Instruction count for the Check All & Port module when the false path is followed.  |
| TSTOP                 | Set by user   | <i>TSTOP</i> is used to tell the simulation when to stop. If all of the incoming packet headers are not read from the file and TSTOP has been reached, the simulation stops in midstream. |
| Global Seed           | Set by user   | <i>Global Seed</i> is used specifically in the simulations where the IP packet headers were randomly generated. For this simulation effort three different seeds were utilized.           |

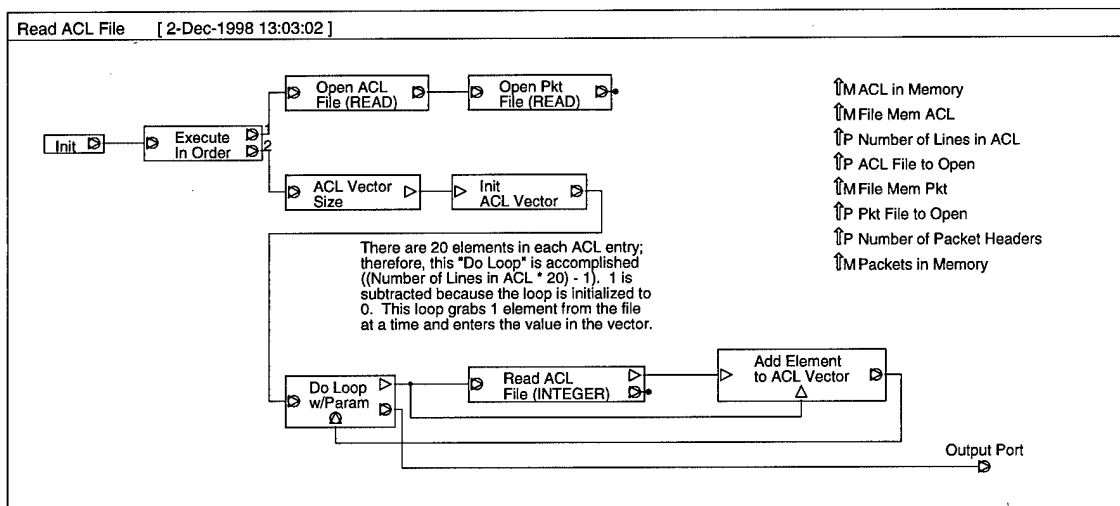
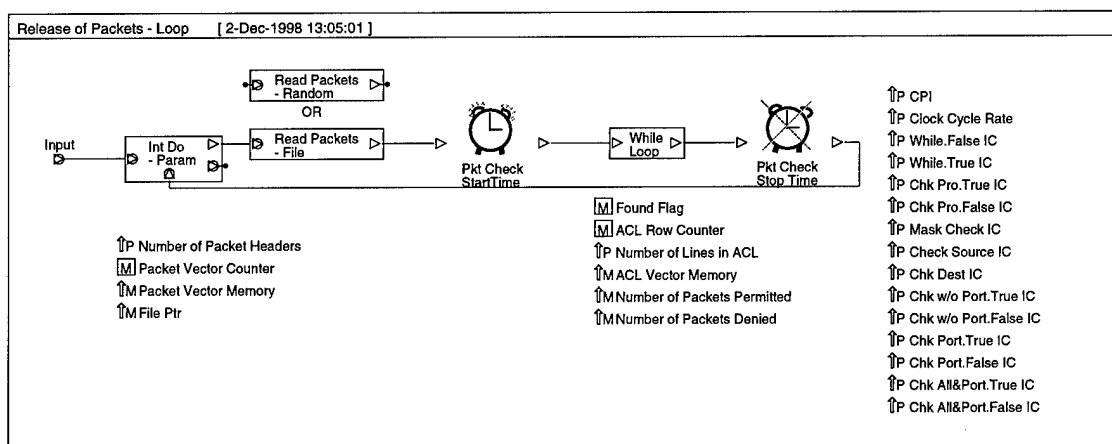


Figure 13. Read ACL File Module

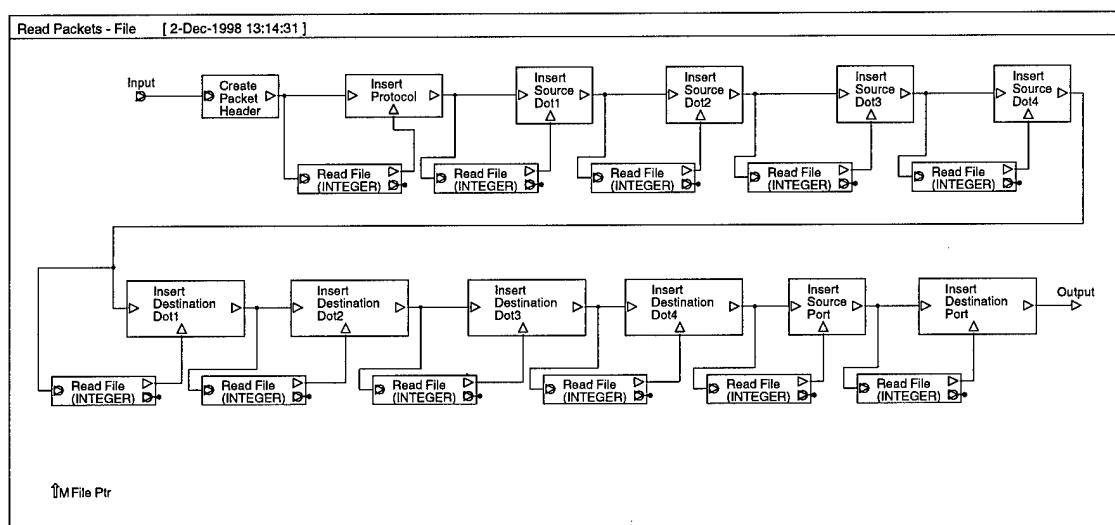
The *Read ACL File* module (Figure 13) starts by initializing the ACL Model; this step involves verifying all of the Designer modules necessary for a successful execution are present. Also accomplished during the initialization phase is the setting of all variables. The model proceeds to open the Access Control List (ACL) and IP Packet Header files for reading. Once the ACL file is opened, each element in the file is read and placed into the *ACL in Memory* vector. As necessary during each simulation, blocks will access the ACL values stored in the vector. This approach was preferred over continuous I/O disk operations that could conceivably slow the simulation. In addition, having the entire ACL in memory more accurately depicts how an ACL is stored in a router. Upon completion of this module, control of the simulation returns to the ACL Model Top Level System component, where the next system call activates the *Release of Packets – Loop* module.



**Figure 14. Release of Packets - Loop Module**

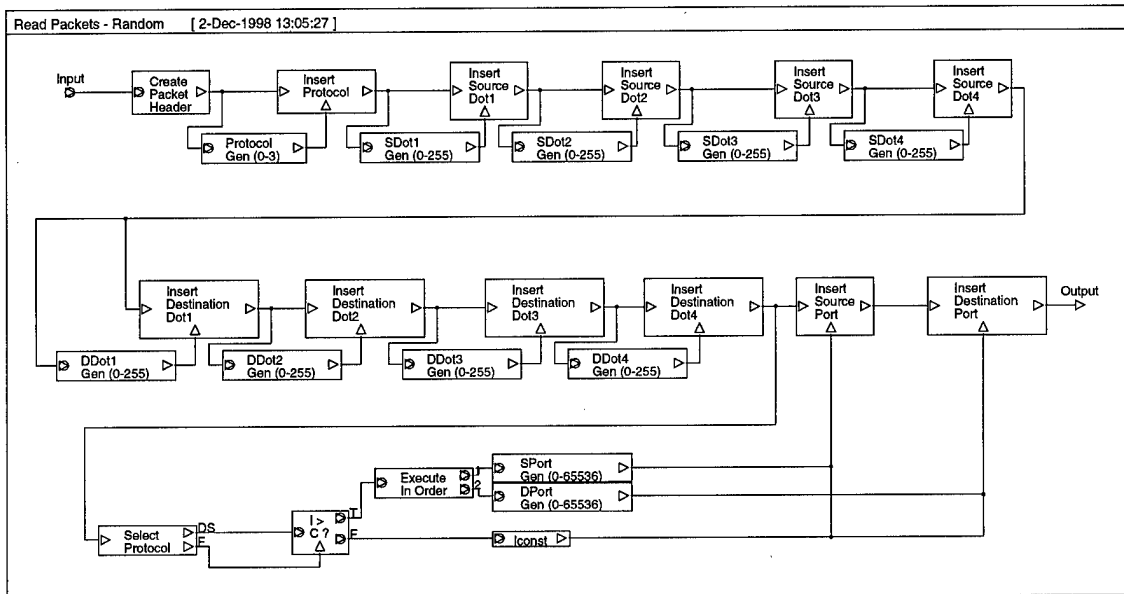
Up to this point all of the steps accomplished were part of the simulation initialization. There are two variations of the *Release of Packets - Loop* module (Figure 14). The figure above is designed to read packet header data from a file specified by the user. The second variation of the module generates random packet header data, based on

a user specified *Global Seed*. The approach of reading the packets from a file was necessitated in this instance due to a limitation in the size of vector allowed by Designer. Attempts were made to place the entire test data sets (each file consisted of 11,443 lines, a line was composed of 11 elements each) into a vector structure; however, Designer could not manage a structure of such magnitude. Therefore, the simulation was designed to read one line at a time from the file and placed them into a data structure called *Packet Header*. Once the *Packet Header* data structure was populated it was released and thus triggered the *Pkt Check Start Time* block, which entered the current simulation time into the appropriate field in the data structure. The *Packet Header* is passed to the *While Loop* module. Upon exiting the *While Loop* module the current simulation time is entered into the appropriate field in the *Packet Header* data structure.



**Figure 15. Read Packets - File Module**

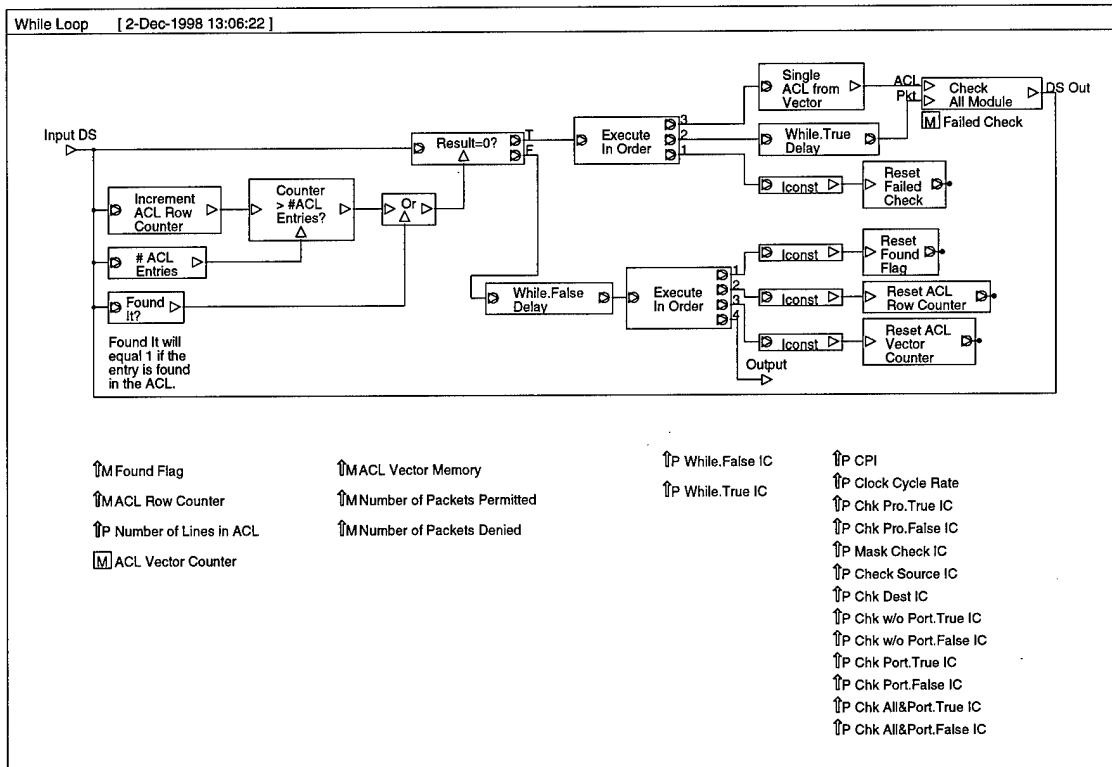
The *Read Packets – File* module (Figure 15) is responsible for reading the IP packet header input file and placing them into the *Packet Header* data structure. Section 4.3.1.2 contains a copy of the *Packet Header* data structure definition, and Appendix B contains a sample entry from the input file.



**Figure 16. Read Packets - Random Module**

The *Read Packets – Random* module (Figure 16) is designed to generate random inputs into the *Packet Header* data structure. Each random generation block was given a range from which values could be drawn. If the protocol was of the type TCP or UDP it was necessary to generate source port and destination port values, otherwise the last two entries were filled with zero (0).





**Figure 17. While Loop Module**

The *While Loop* module (Figure 17) is responsible for identifying the status of a packet header passing through the packet validation component. The status of a packet header passing through the *While Loop* module is maintained by the *Found Flag* variable. The *Found Flag* when set to one (1) identifies to the program an entry has been found in the ACL, get the next packet header. In the event the *Found Flag* is zero (0), the packet header is scanned against the next ACL entry. When a packet arrives at the router interface, it is identified as a new packet and it must be scanned against the assigned ACL. The following rule is used to determine the next operation in the ACL Model.

***WHILE NOT* ACL Row Counter > # ACL Entries *OR* Found Flag = 1 DO**

As long as the above condition is true, an entry from the *ACL in Memory* vector is extracted and placed in the *Single ACL Entry* data structure, this action is accomplished in

Figure 18. At this point the *Packet Header* and *Single ACL Entry* data structures have been initialized with the proper data, and the structures begin to traverse the appropriate path as designed in the model.

In the case where the end of an ACL has been encountered or a match was found, the *While.False Delay* would be applied. Throughout this appendix, delay is dealt with in terms of seconds. The instruction count for the *While.False IC* (5) variable is computed in the following manner\*:

| Instruction                                | Result of Instruction  |
|--|--|
| <b>add</b> ACL counter + 1                 | Increment the ACL line counter   |
| <b>slt</b> Number of ACL Entries < Counter | If the number of ACL entries is less than counter, then set register to 1                            |
| <b>or</b> SLT Result, Found It             | Result = 1, if Number of ACL Entries < Counter OR Found It = 1                                       |
| <b>beq</b> Result of OR = 1                | Jump to <i>Check All Module</i> (Figure 19)  |
| <b>jump</b> to Release of Packets module   | Jump out of the <i>While Loop</i> – Clock stopped.<br>At this point all variables are reinitialized. |

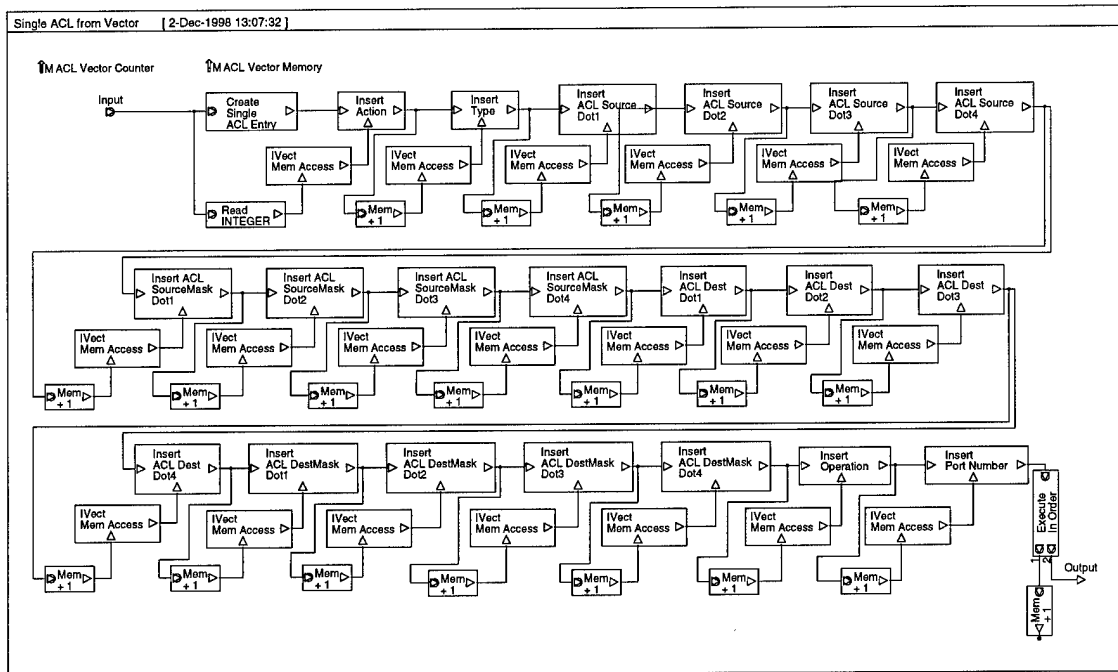
The *While.True IC* (4) variable utilizes only the first four instructions of the set above, since it needs to check the incoming *Packet Header* protocol with the next *Single ACL Entry* type.

The ACL Model provides two viable paths the two data structures can traverse; the first check in the program identifies the protocol type of the incoming packet. If the packet is of the type TCP or UDP, the simulation routes the data structures to the *Check All & Port* module (Figure 20). The *Check All & Port* module is comprised of four sub-modules: *Check Protocol*, *Check Source*, *Check Destination* and *Check Port*. If the incoming packet is of the type IP or ICMP, the data structures are routed to the *Check w/o*

---

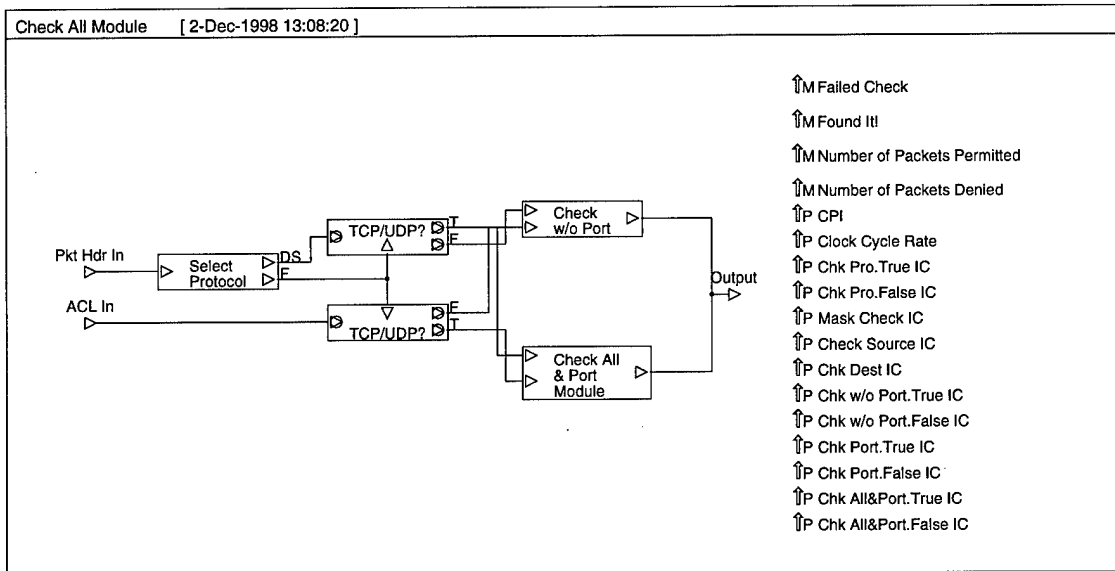
\* Note: The instructions used in calculating the module Instruction Count values are in pseudocode.

*Port* Module (Figure 26); this module consists of the previously listed sub-modules except *Check Port*.



**Figure 18. Single ACL from Vector Module**

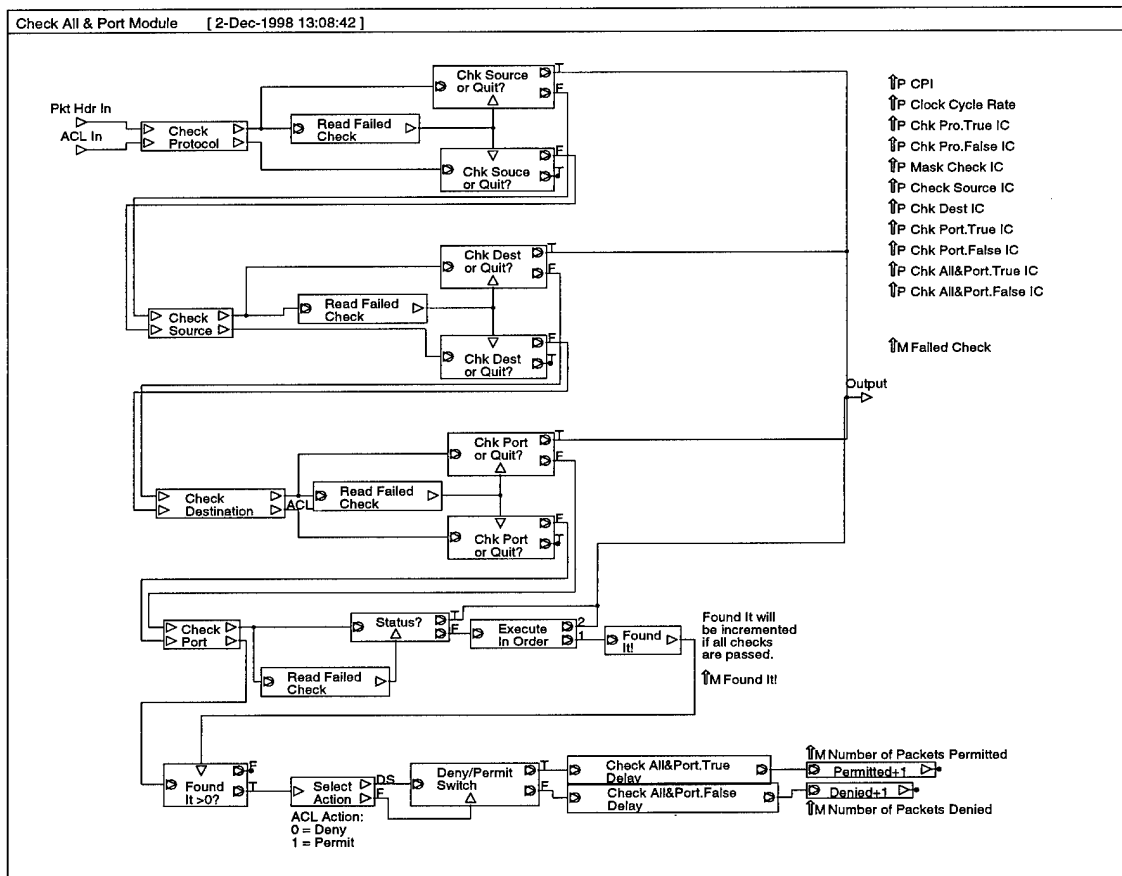
The *Single ACL from Vector* module is designed to sequentially read values (in groups of 20) from the *ACL in Memory* vector and place them into the *Single ACL Entry* data structure. Section 4.3.1.1 contains a copy of the *Single ACL Entry* data structure definition, and Appendix B contains a sample entry from the input file. The *ACL Vector Counter* is an integer variable used to maintain the proper position within the vector.



**Figure 19. Check All Module**

The Check All Module (Figure 19) does not contribute to the instruction count. This module's sole purpose is to identify the protocol value stored in the *Packet Header* data structure and route the *Packet Header* and *Single ACL Entry* data structures to the appropriate module. In the case where the IP packet is of type TCP or UDP protocol, the data structures are forwarded to the Check All & Port module (Figure 20). The IP or ICMP protocol packets are directed to the Check w/o Port module (Figure 26).

The following figures are in the order they are encountered if the TCP/UDP path is followed and each check is successful.

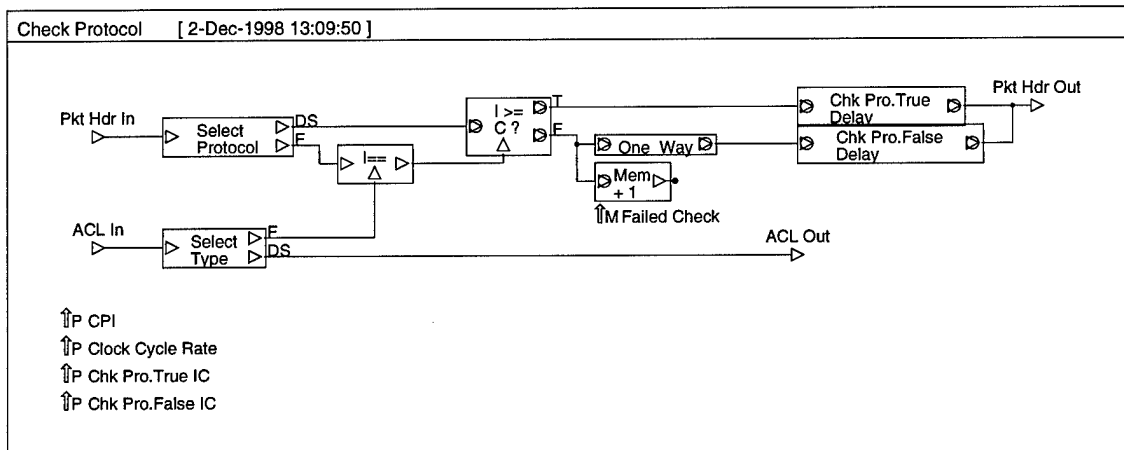


**Figure 20. Check All & Port Module (TCP/UDP)**

In the *Check All & Port Module* the *Single ACL Entry* and *Packet Header* data structure are routed to the *Check Protocol* module (Figure 21). As the two data structures emerge from *Check Protocol*, the *Read Failed Check* block is triggered. If the *Failed Check* variable is greater than zero (0), control of the simulation is returned to the *While Loop* module, where the next ACL entry is obtained and the process is started or a new IP packet header is received. This basic flow is repeated for each of the remaining “Check” modules.

The methodology used to determine the instruction counts for the *Check All & Port.False* (2) and *Check All & Port.True* (3) variables are as follows:

| Instruction                      | Result of Instruction                 |
|----------------------------------|---------------------------------------|
| <b>beq</b> Select Action = 0     | Jump to DENY instruction              |
| <b>beq</b> Select Action = 1     | Jump to PERMIT instruction            |
| DENY <b>jump</b> to While Loop   | Packet is discarded                   |
| PERMIT <b>jump</b> to While Loop | Packet allowed to flow through router |

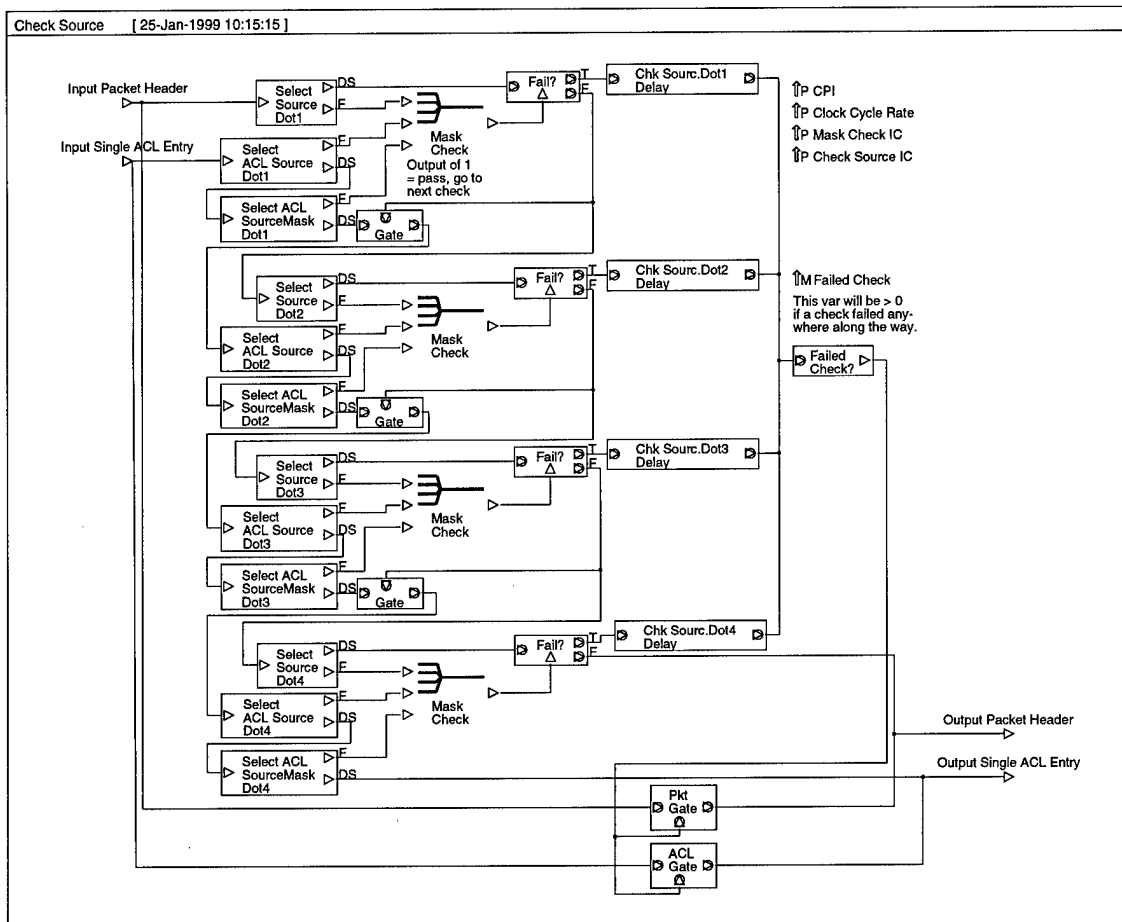


**Figure 21. Check Protocol Module**

The first comparison between the incoming packet and the access control list occurs within the *Check Protocol* module (Figure 21). The value associated with the *Protocol* field in the *Packet Header* data structure is compared to the value in the *Type* field of the *Single ACL Entry* data structure. If the two values are not equal the *Failed Check* flag is incremented and the control of the simulation returns to the *Check All and Port* module. When the protocol values are equal, the two data structures are passed to the *Check Source* module.

The methodology used to determine the instruction count for the *Check Pro.False* (2) and *Check Pro.True* (1) variable is as follows:

| Instruction   | Result of Instruction                      |
|---|--|
| <b>beq</b> Packet Header.Protocol = Single ACL Entry.Type | Branch to the Check Source.Mask Check      |
| <b>jump</b> to While Loop                                 | Protocol check failed return to While Loop |



**Figure 22. Check Source Module**

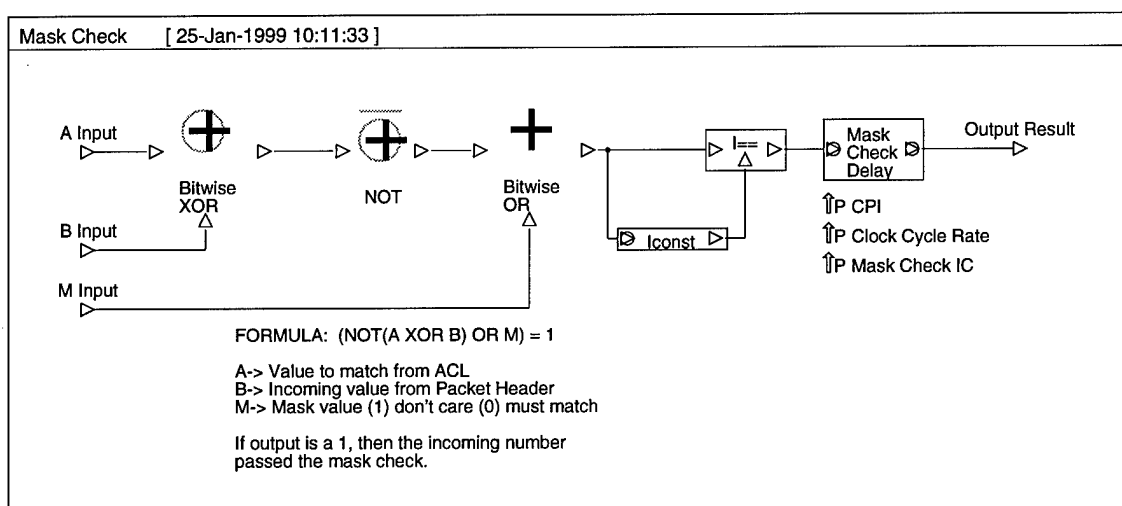
Upon a successful protocol check, the *Check Source* module (Figure 22) is the next hurdle the incoming packet must overcome. Internet Protocol source and destination addresses are 32-bits in length, for the purpose of this modeling effort each address was decomposed into four 8-bit chunks (e.g., 129.92.1.2). In the example, the value 129 is given the generic name “Source Dot 1” (Source can be replaced by Destination if the address is a Destination IP address); likewise, 92 is referred to as “Source Dot 2”.

As a packet arrives at the *Check Source* module, the “Source Dot 1” value in the data structure is used as an input variable to the *Mask Check* module (Figure 23). Similarly the “ACL Source Dot 1” and “ACL SourceMask Dot 1” from the *Single ACL Entry* data

structure are used as inputs to the *Mask Check* module (Figure 23). Upon a failed mask check, the *Failed Check* flag is incremented and the control of the simulation returns to the *Check All & Port* module. If the “Source Dot 1” value passes the mask check, the “Source Dot 2” fields and mask value is extracted from the applicable data structures, and another mask check is performed. There are four “Dot” checks, if each check is successful the two data structures are forwarded to the *Check Destination* module (Figure 24).

The Check Source instruction count (1) is simply one instruction, shown below. The delay blocks are taken in the event one of the Mask Check modules fail.

| Instruction               | Result of Instruction                                      |
|---------------------------|--|
| <b>jump</b> to While Loop | Return control of the simulation to the While Loop module. |



**Figure 23. Mask Check Module**

Essentially the *ACL Mask Check* module (Figure 23) is the heart of the ACL Model. All of the Designer modules explained thus far are required to control the flow of the



packets through the router. The *Mask Check* module operates on the bit level, comparing two numbers based on a “mask” value.

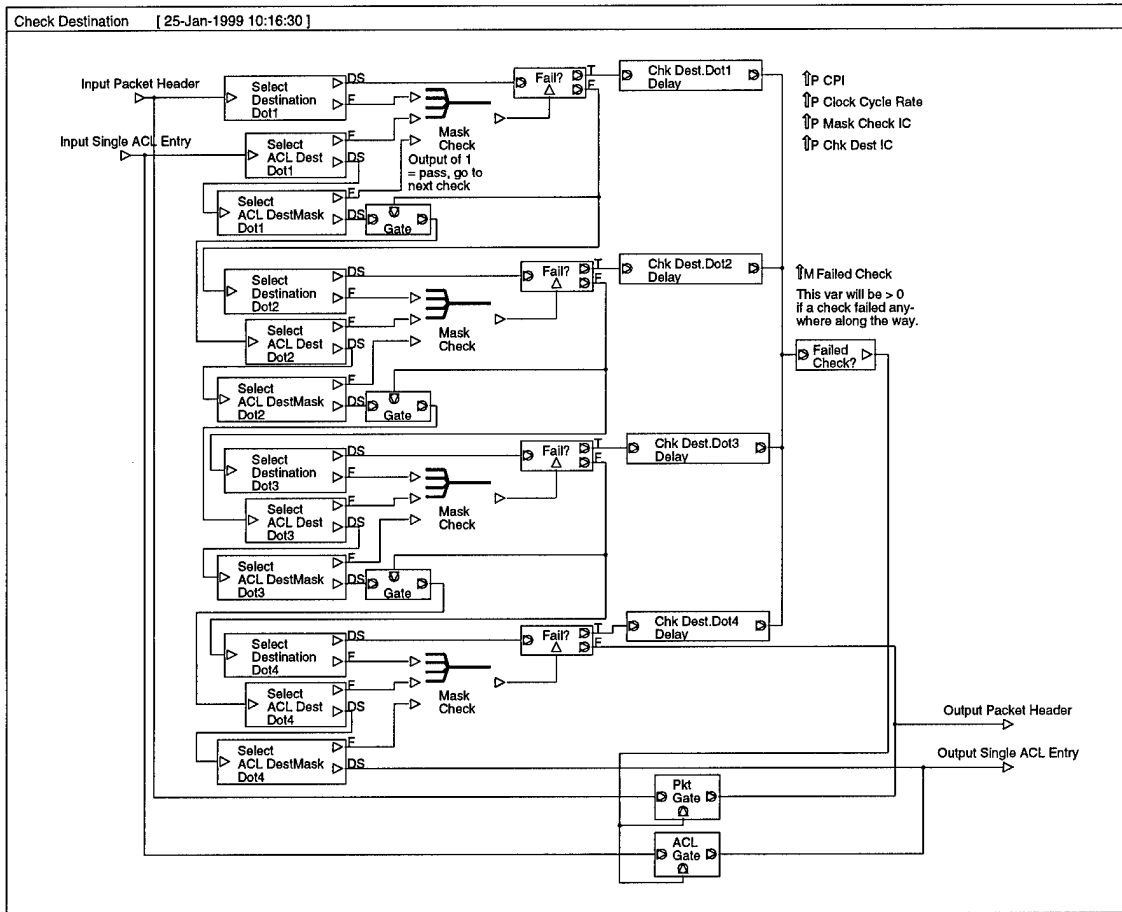
The *Mask Check* module was created by using logical bitwise operators, such as XOR and OR. The formula used to calculate whether an incoming packet matches an entry in the ACL can be seen below.

$$(\text{not}(A \text{ xor } B) \text{ or } M) = 1 \quad (3)$$

The value associated with A represents the number to match from the ACL. Variable B represents a portion of the incoming source or destination address of the *Packet Header* data structure. M represents the mask value from the appropriate entry in the access control list. If the result of the formula yields a 1, the incoming bit passed the check. The result of each check is forwarded to the module responsible for initiating the check.

Computation of the instruction count for the *Mask Check* module is based on four separate MIPS instructions. The instruction set is as follows:

| Instruction        | Result of Instruction                   |
|--------------------|---|
| <b>xor</b> A B     | Resultant value stored in C             |
| <b>xor</b> c 255   | Resultant value stored in D             |
| <b>or</b> D M      | Resultant value stored in E             |
| <b>beq</b> E = 255 | Proceed to next Check Source.Mask Check |



**Figure 24. Check Destination Module**

The *Check Destination* module (Figure 24) is identical in construct as the *Check Source* module, except that the destination addresses and mask fields are used. All fields are forwarded to the *Mask Check* module for validation.

If part of the destination address fails any of the *Mask Check* modules, the *Failed Check* flag is incremented. In the event the *Failed Check* variable is greater than zero, the control of the simulation is returned to the *Check All & Port* module.

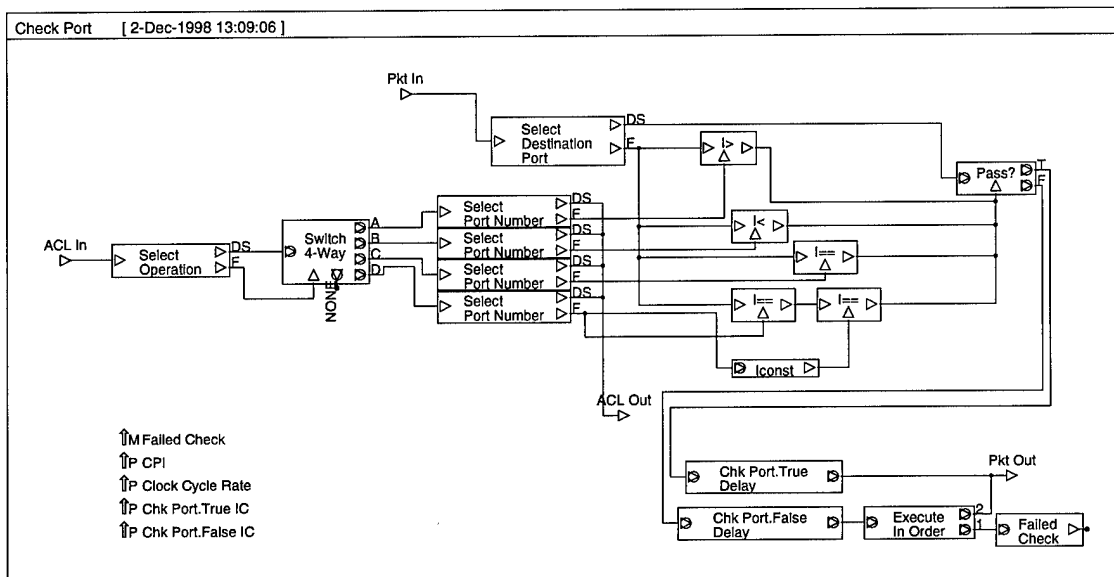
The *Check Destination* instruction count (1) is simply one instruction, shown below. The delay blocks are taken in the event one of the *Mask Check* modules fail.

Instruction

Result of Instruction

**jump** to While Loop

Return to *Check All & Port* module.



**Figure 25. Check Port Module**

Each entry in an ACL, if it is specifically designed to filter TCP or UDP packets, will have two extra fields – *Operation* and *Port Number*. The operator field can take on four different forms, Greater Than, Less Than, Equal To, and Not Equal To. The ACL administrator can specify, “I do not want to accept TCP or UDP packets from source X.X.X.X to destination X.X.X.X, with a destination port address less than 601.” The *Check Port* module (Figure 25) accomplishes the task of determining if the incoming destination port address meets the criteria prescribed by the ACL entry being scanned.

At this point, if the TCP or UDP packet has passed all checks the data structures are returned to the *Check All & Port* module, where the next step is to determine the appropriate action to be accomplished in regards to the packet. Each ACL entry has an *Action* field, permit or deny, based on the value in this field one of two variables is incremented – *Number of Packets Permitted* or *Number of Packets Denied*. In a router,

packets passing all checks would be allowed to continue through the network if the *Action* field is set to “permit”, while all others would be denied.

Dependent upon the path taken by the data structure, based upon the *Single ACL Entry.Operation* field, one MIPS instruction is executed. For example if the *Operation* was “Equal” the following instruction set would be applied to the *Check Port.True* (1) and *Check Port.False* (2) delay blocks. All operations in this research effort require a single instruction to execute.

| Instruction  | Result of Instruction  |
|--|--|
| <b>beq</b> Packet Header.Destination Port =<br>Single ACL Entry.Port Number<br><b>jump</b> to While Loop | If the result is equal, branch to ACTION of the<br>Check All & Port or Check w/o Port module<br>Return control to the While Loop |



The methodology used to determine the instruction count for the *Check w/o*

*Port.False* (2) and *Check w/o Port.True* (3) variable is as follows:

| Instruction                      | Result of Instruction                 |
|----------------------------------|---------------------------------------|
| <b>beq</b> Select Action = 0     | Jump to DENY instruction              |
| <b>beq</b> Select Action = 1     | Jump to PERMIT instruction            |
| DENY <b>jump</b> to While Loop   | Packet is discarded                   |
| PERMIT <b>jump</b> to While Loop | Packet allowed to flow through router |

## C.2 Instruction Count Variability

The total instruction count calculated for each iteration of the *While Loop* module is dependent upon the composition of both the ACL entry and the incoming packet header. The following examples provide some insight as to the variability of the instruction count from one iteration of the *While Loop* module to the next.

An example of the shortest possible instruction count achieved for a single ACL entry and a single packet header can be shown with the following entry from an ACL and the pertinent data of an incoming packet header.

### ACL Entry

```
access-list 101 permit ip 0.0.0.0 255.255.255.255 0.0.0.0 255.255.255.255
```

### Incoming Packet Header

```
Protocol          icmp
Source Address    109.90.2.1
Destination Address 129.92.126.2
```

When an ACL entry is configured for a specific protocol, such as IP in the above example, an incoming packet header of the type ICMP would fail the protocol check. From the time this packet enters the *While Loop* until it exits, a total of 10 instructions have been executed.

An example of the longest possible instruction count achieved for a single ACL entry and a single packet header can be shown with the following entry from an ACL and the pertinent data of an incoming packet header.

#### ACL Entry

```
Access-list 101 permit tcp 0.0.0.0 255.255.255.255 0.0.0.0 255.255.255.255 eq 25
```

#### Incoming Packet Header

|                     |              |
|---------------------|--------------|
| Protocol            | tcp          |
| Source Address      | 109.90.2.1   |
| Destination Address | 129.92.126.2 |
| Source Port         | 80           |
| Destination Port    | 35           |

In the above example the ACL entry is designed to permit all TCP packets from any source address destined for any other address if the destination port is equal to 25. The incoming packet header provided above meets all of the criteria exactly, allowing it to pass through the router. From the time this packet enters the *While Loop* until it exits, a total of 46 instructions have been executed.

### **C.3 Summary**

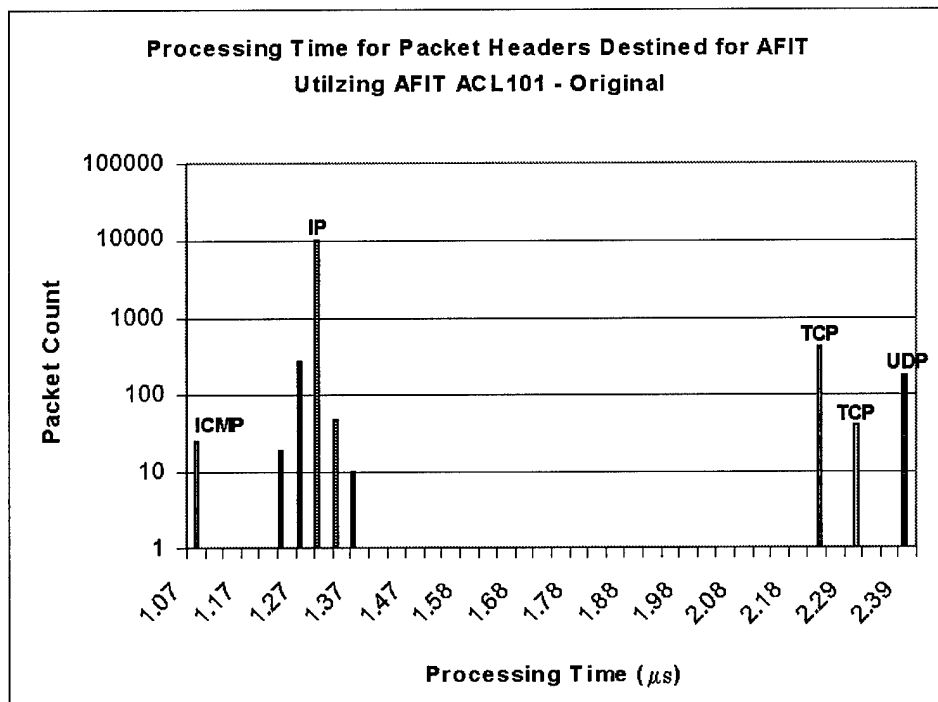
This appendix has detailed the inner components of the ACL Model, as designed using the BONEs Designer tool. Discussed was the flow of the two data structures (Packet Header and Single ACL Entry) through the ACL Model and the steps involved in scanning a packet header against a single entry from an ACL. Throughout this appendix the methodology followed in determining an instruction count value for the various ACL Model modules was explained. The last portion of the appendix addressed the topic of instruction counts, highlighting the variability of the total instruction count from one iteration of the *While Loop* module to the next.

## ***Appendix D. Supplemental Graphs***

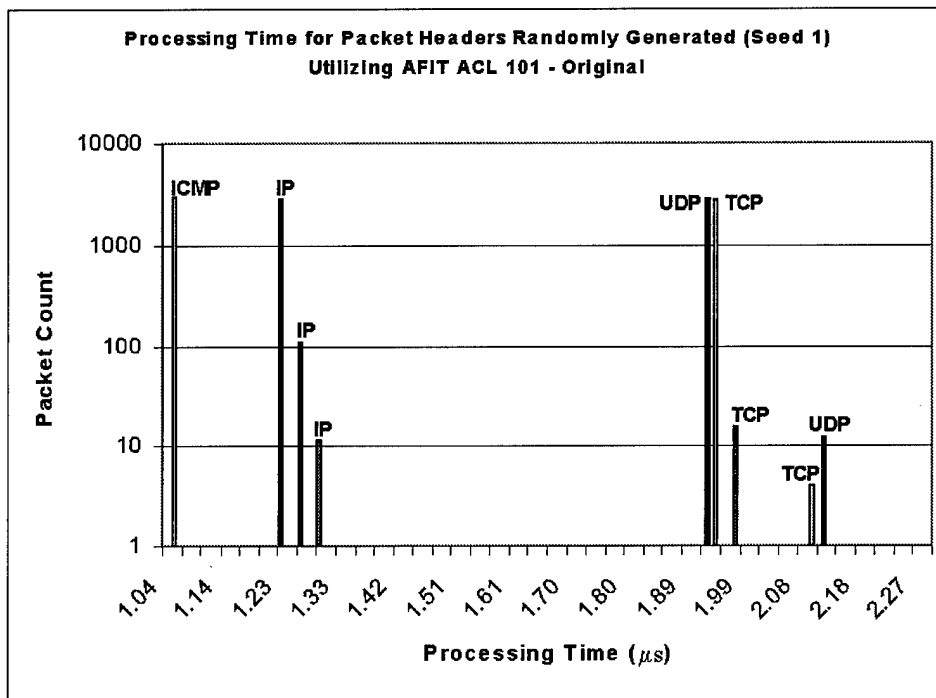
This appendix is designed to provide the reader a full complement of graphs to view while reading Chapter 5. The included graphs encompass the results of each simulation against the original and “optimized” ACLs. Each optimization of an ACL, required five simulations:

1. IP packets destined for a router with the applicable ACL assigned
2. Randomly generated IP packet headers (3 different seeds)
3. IP packets destined for a router with another a different ACL assigned

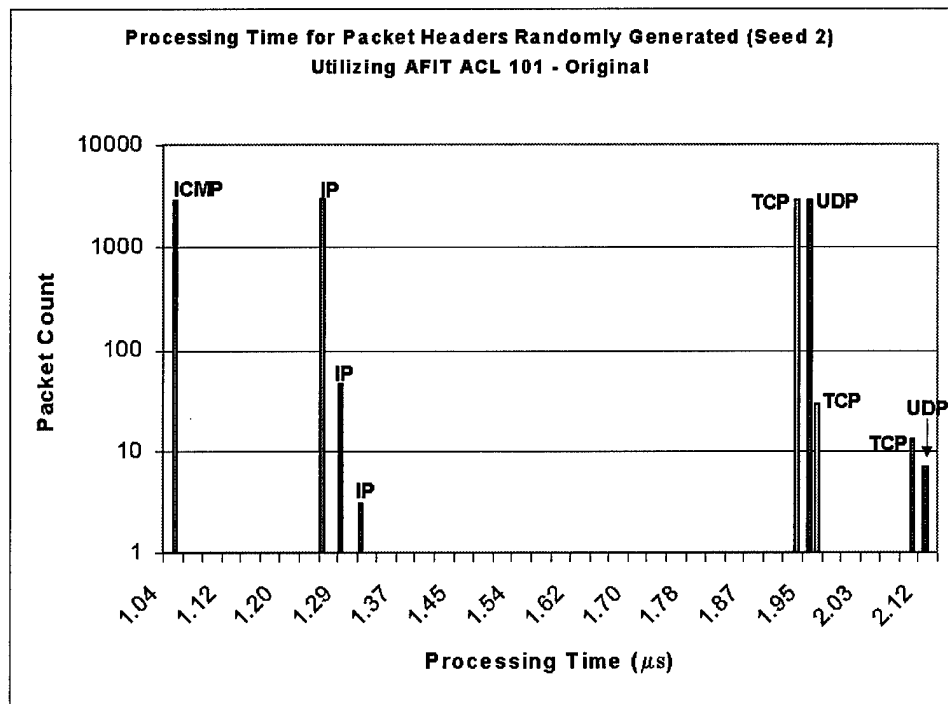




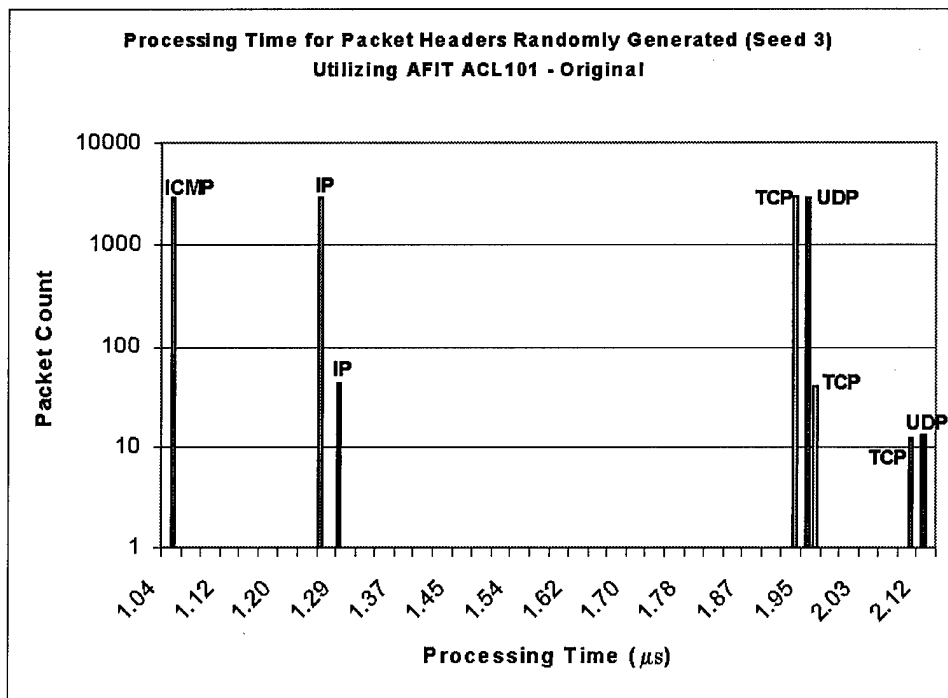
**Figure 27. Processing Time (PT) for AFIT Packets Utilizing AFIT ACL - Original**



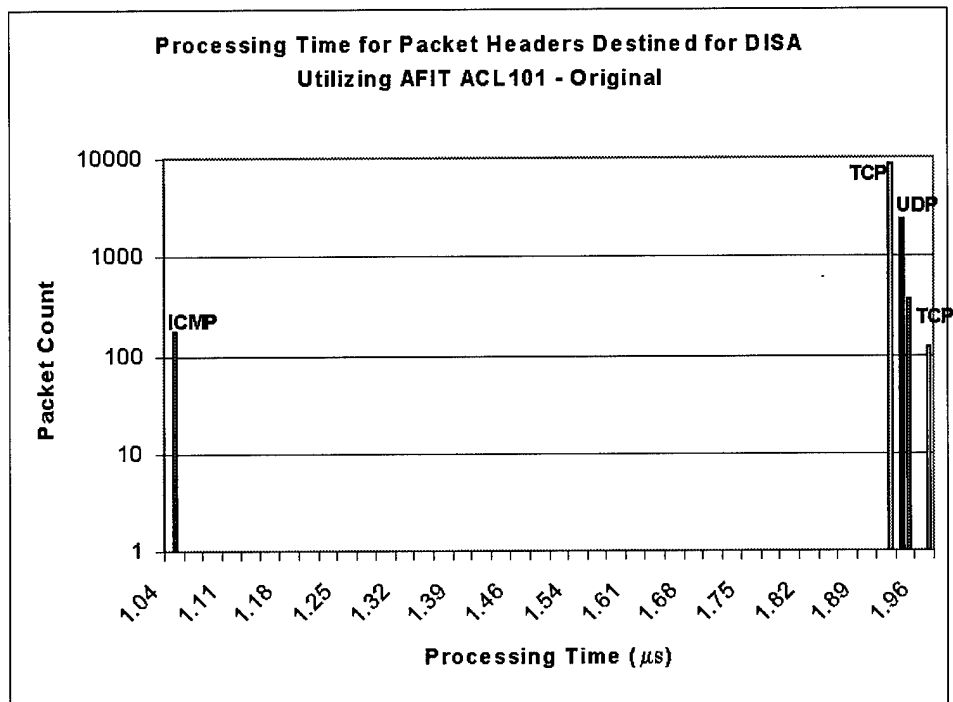
**Figure 28. PT for Random Packets (Seed 1) Utilizing AFIT ACL - Original**



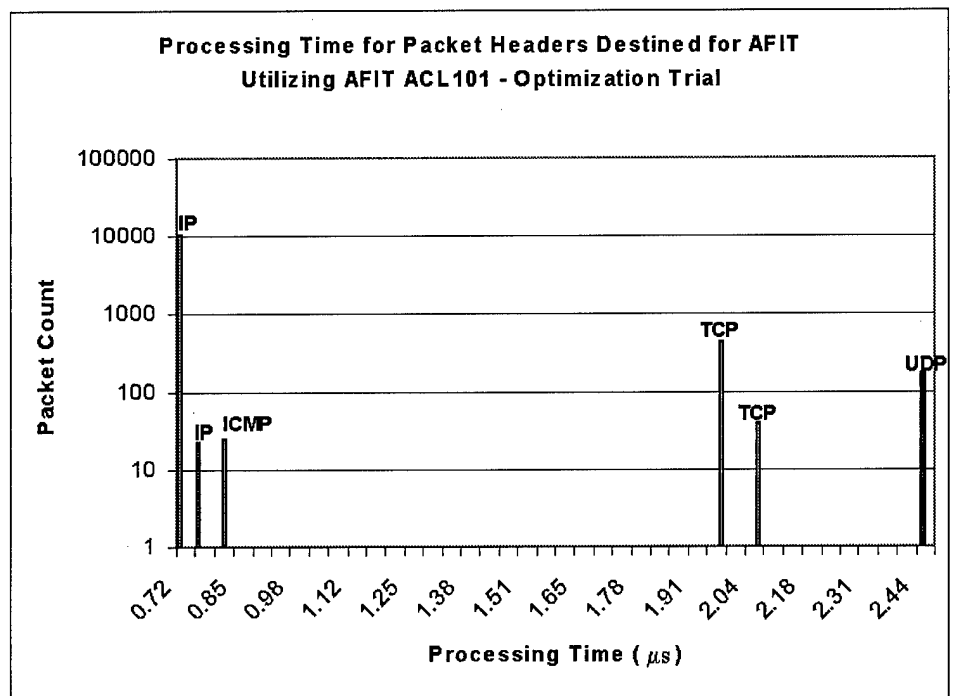
**Figure 29. PT for Random Packets (Seed 2) Utilizing AFIT ACL - Original**



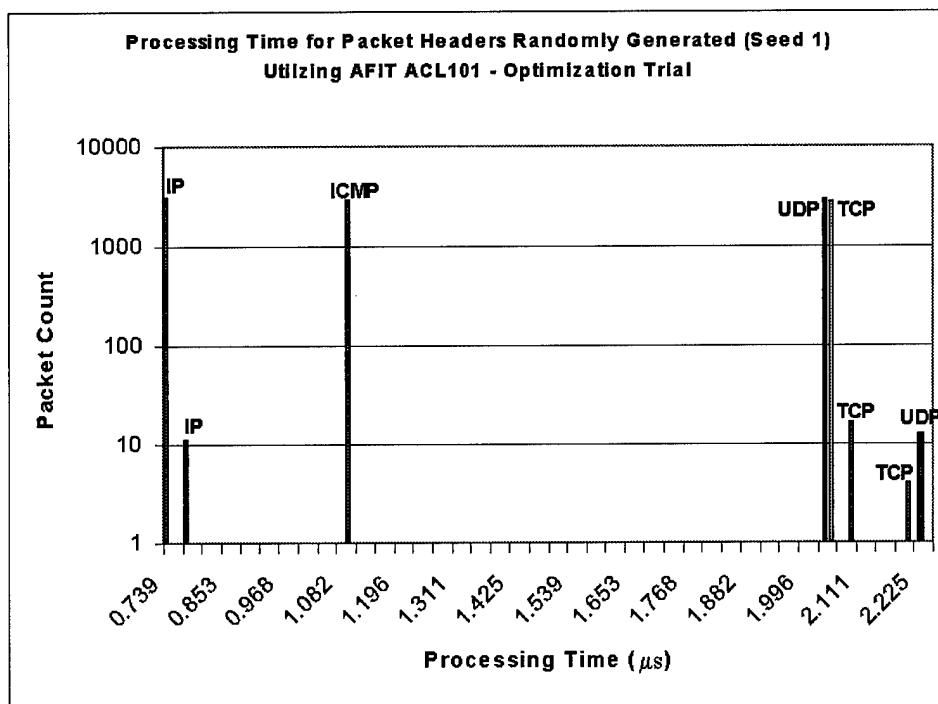
**Figure 30. PT for Random Packets (Seed 3) Utilizing AFIT ACL - Original**



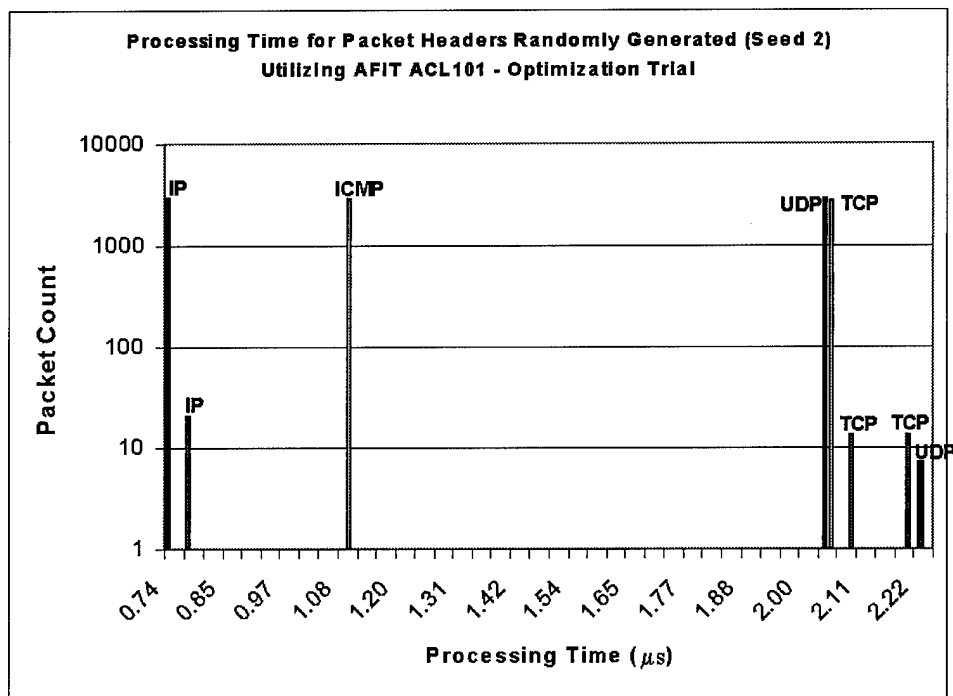
**Figure 31. PT for DISA Packets Utilizing AFIT ACL – Original**



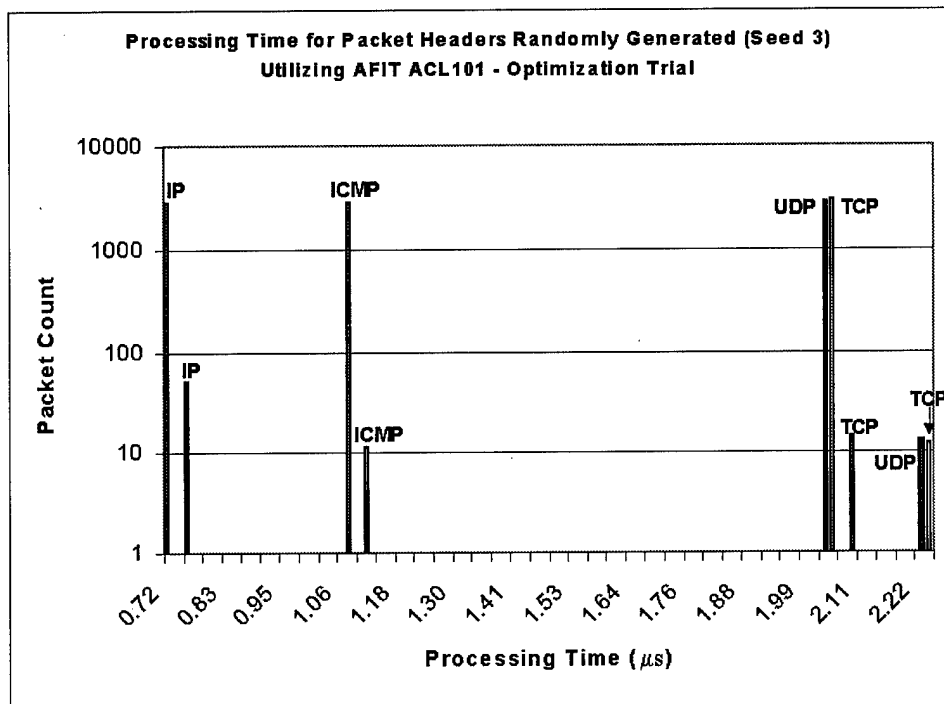
**Figure 32. PT for AFIT Packets Utilizing AFIT ACL – Optimization Trial**



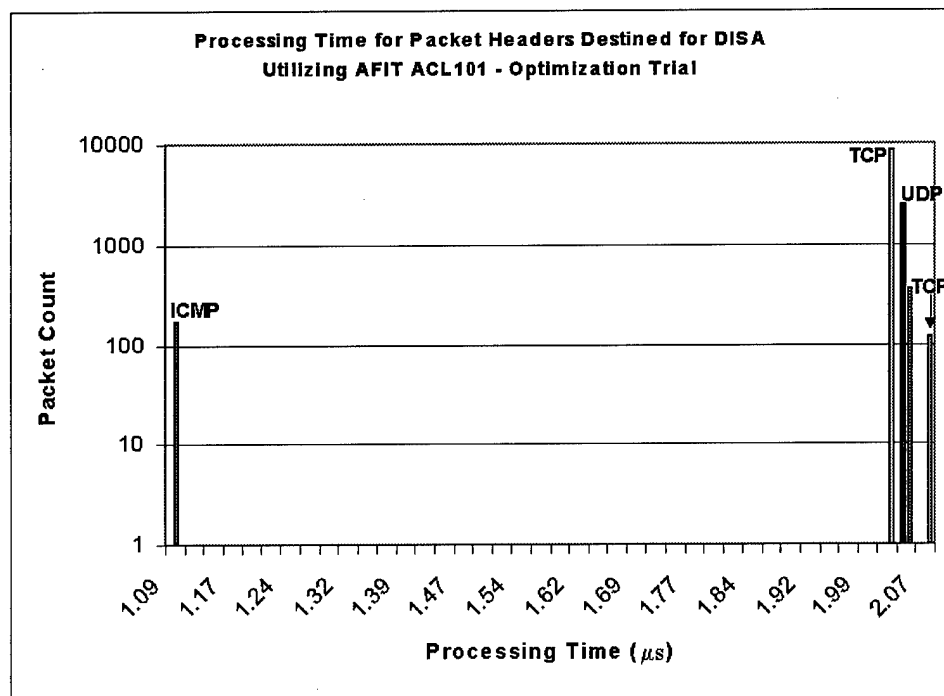
**Figure 33. PT for Random Packets (Seed 1) Utilizing AFIT ACL – Optimization Trial**



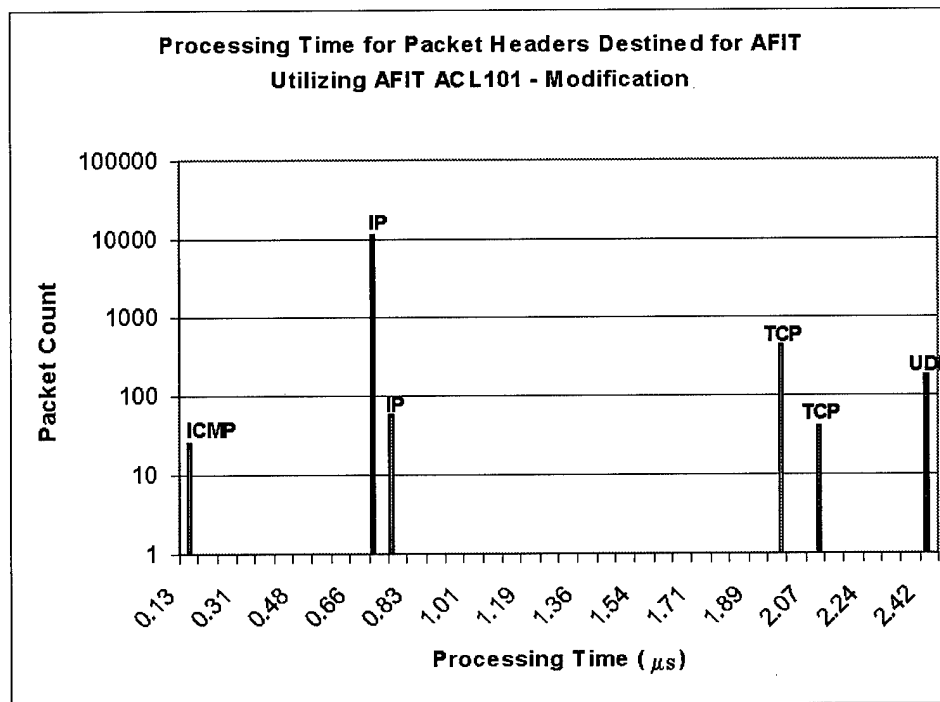
**Figure 34. PT for Random Packets (Seed 2) Utilizing AFIT ACL – Optimization Trial**



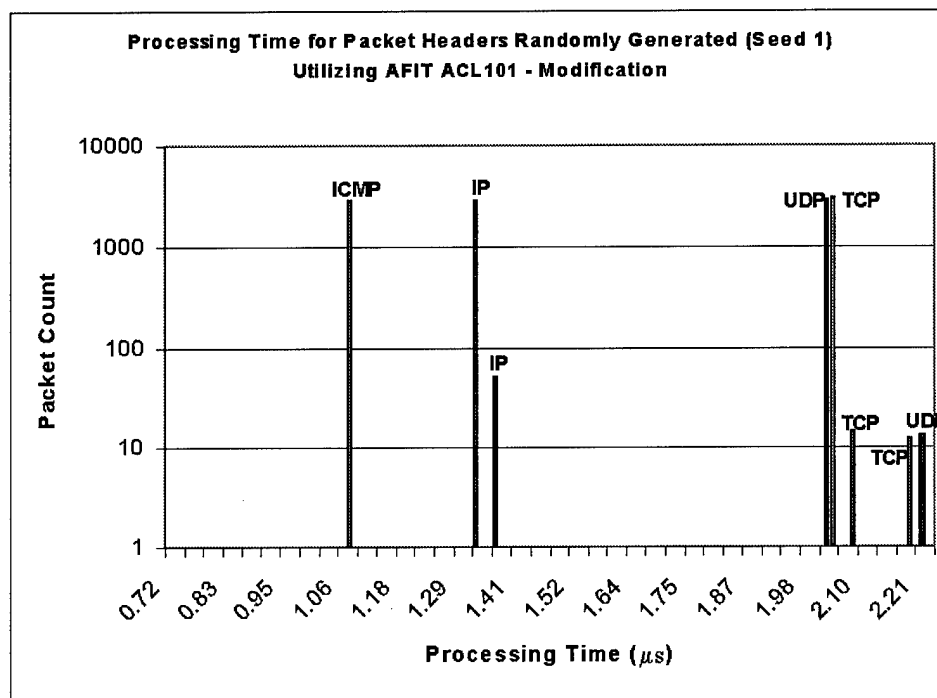
**Figure 35. PT for Random Packets (Seed 3) Utilizing AFIT ACL – Optimization Trial**



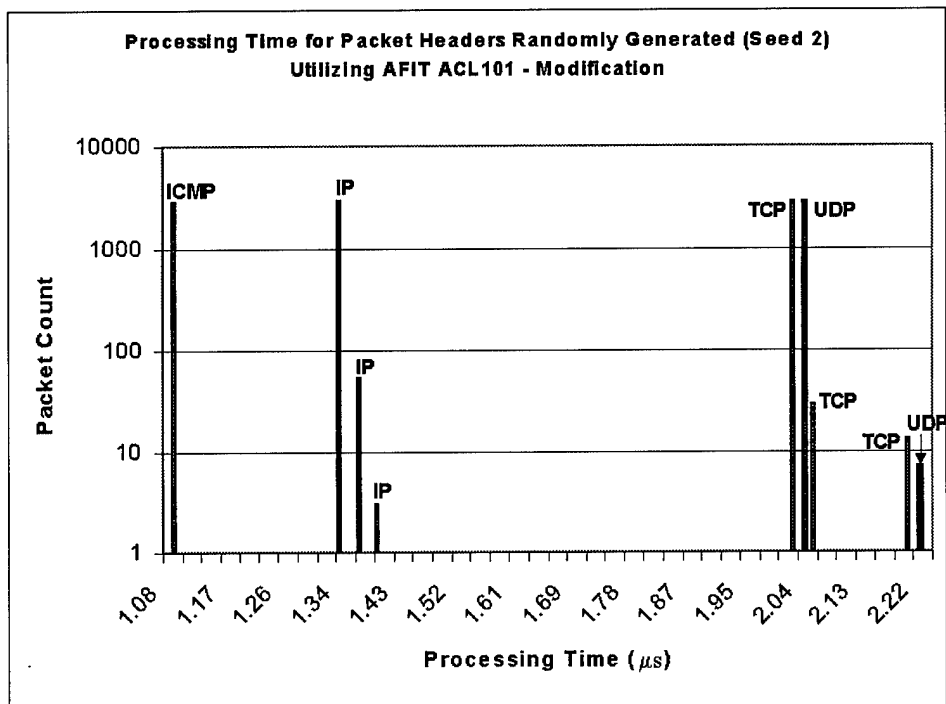
**Figure 36. PT for DISA Packets Utilizing AFIT ACL – Optimization Trial**



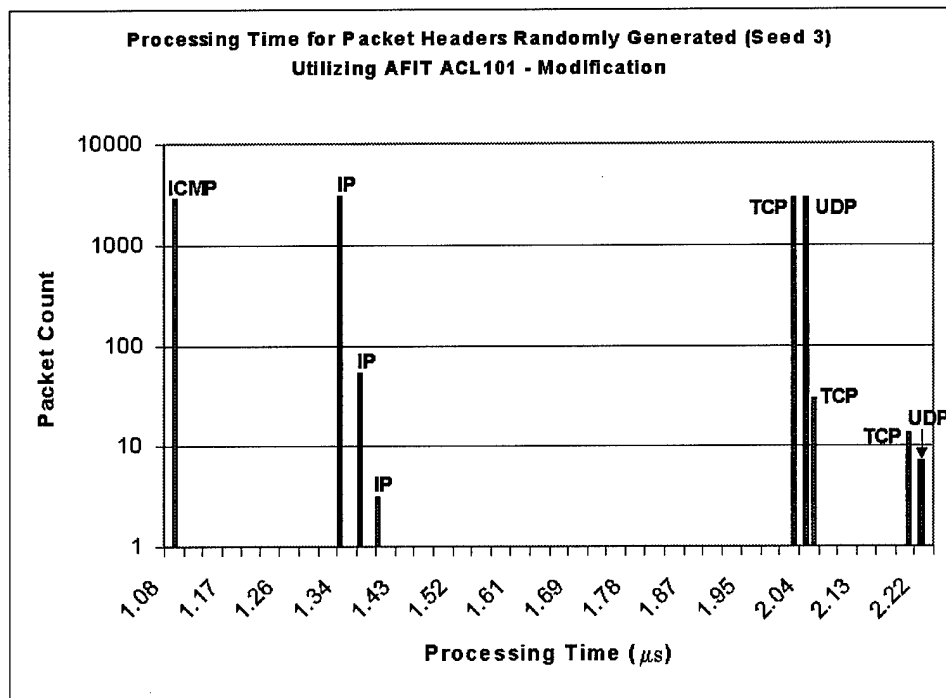
**Figure 37. PT for AFIT Packets Utilizing AFIT ACL – Modification**



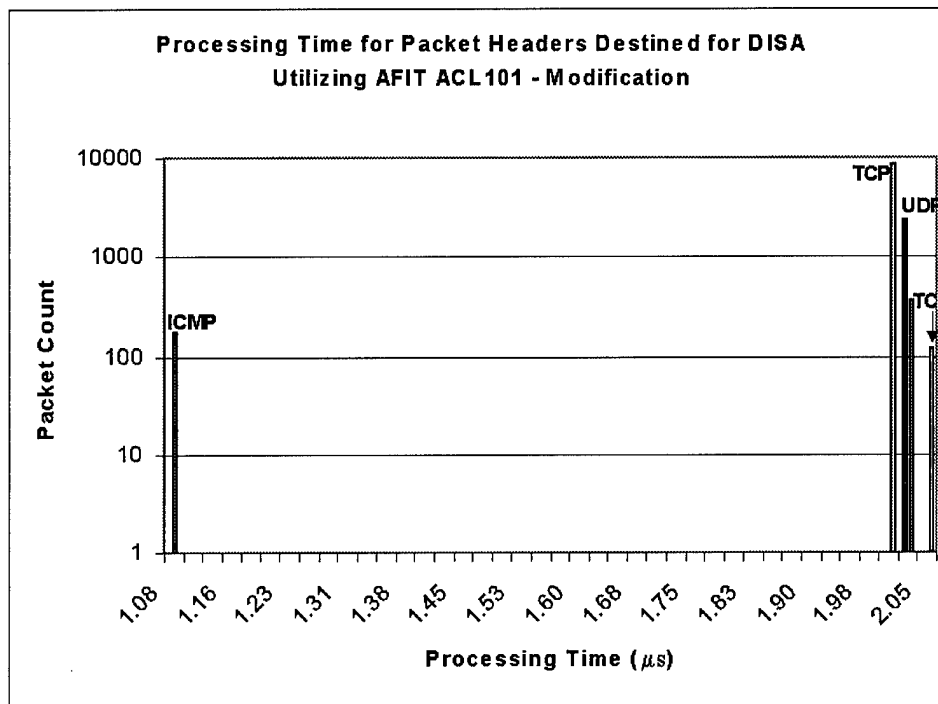
**Figure 38. PT for Random Packets (Seed 1) Utilizing AFIT ACL – Modification**



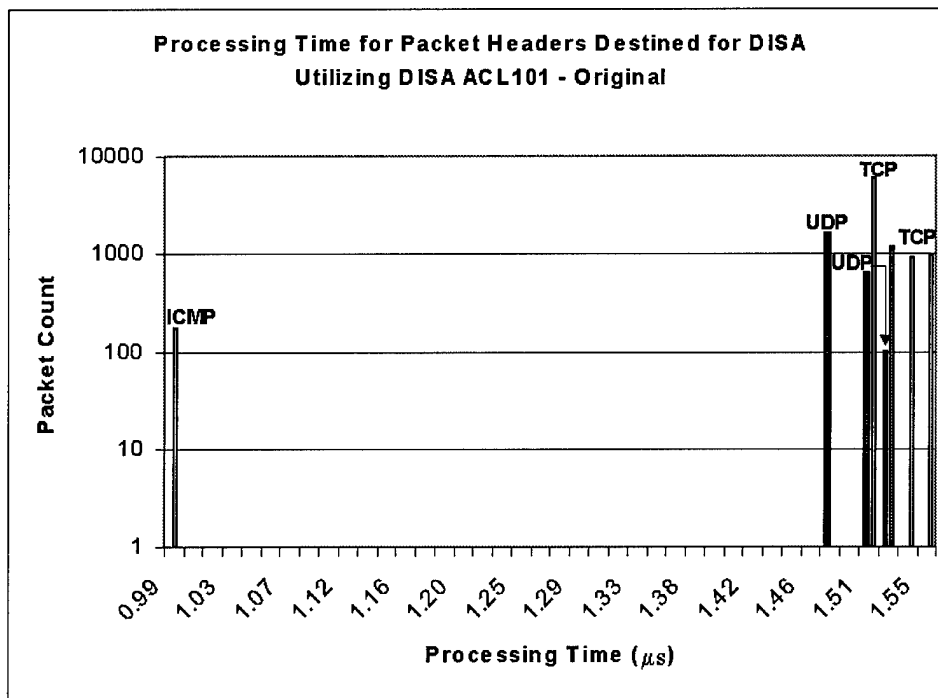
**Figure 39. PT for Random Packets (Seed 2) Utilizing AFIT ACL – Modification**



**Figure 40. PT for Random Packets (Seed 3) Utilizing AFIT ACL – Modification**

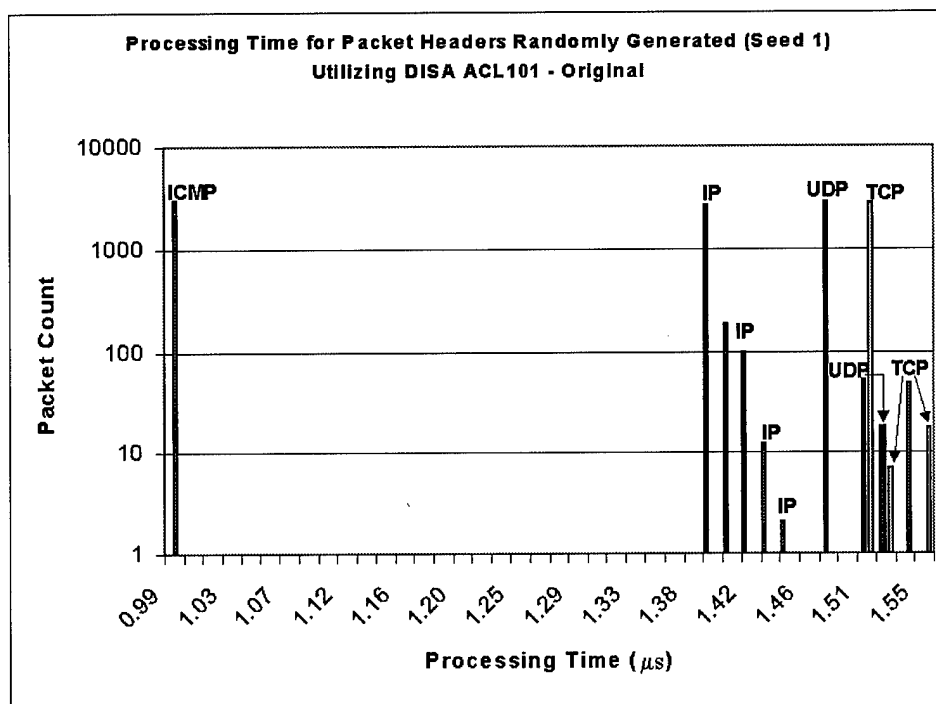


**Figure 41. PT for DISA Packets Utilizing AFIT ACL – Modification**

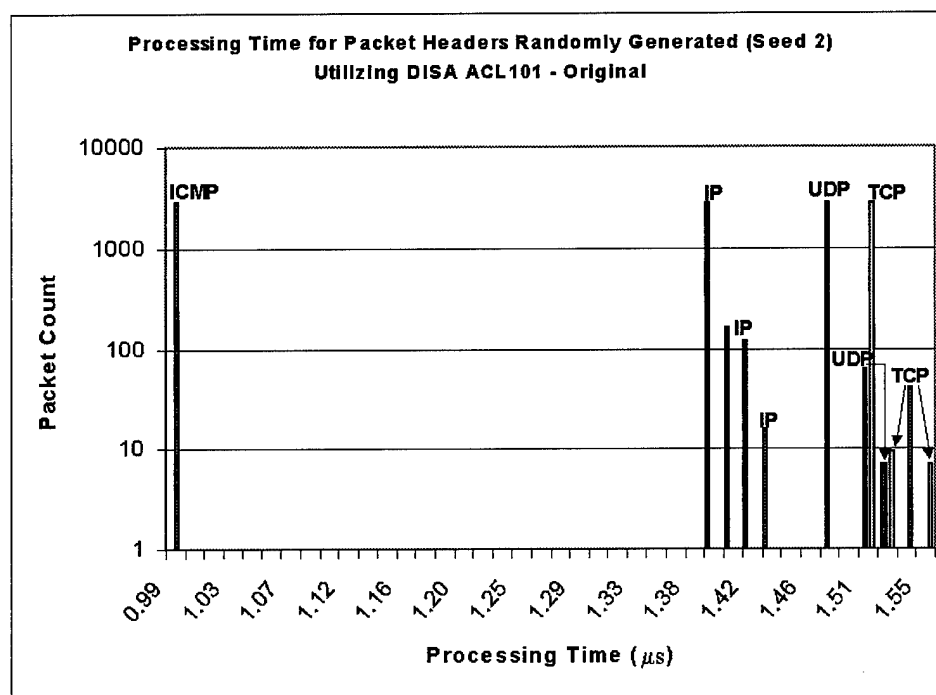


**Figure 42. PT for DISA Packets Utilizing DISA ACL – Original**

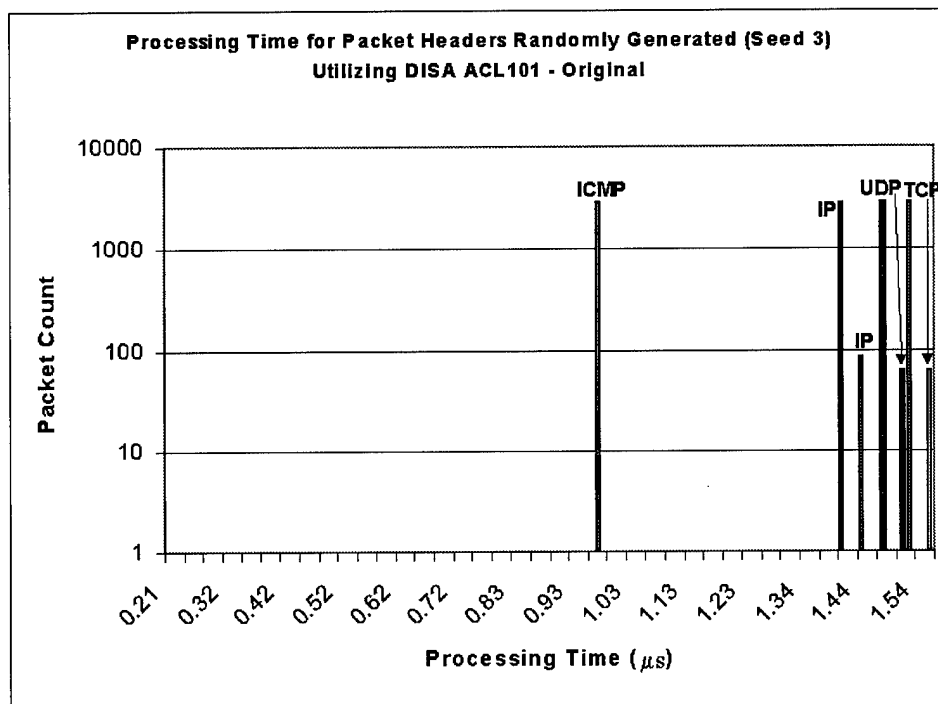




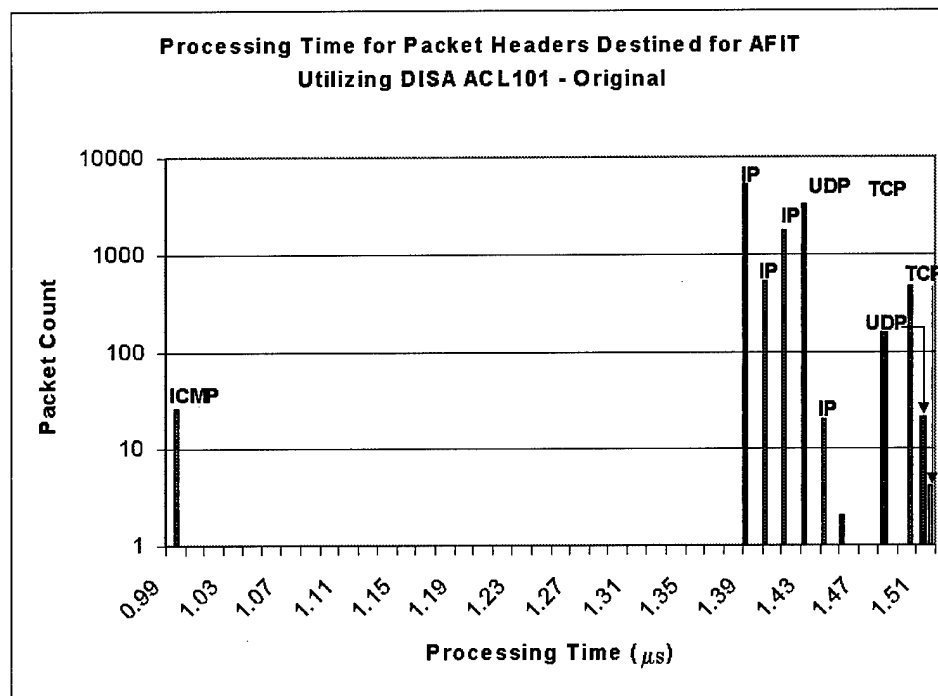
**Figure 43. PT for Random Packets (Seed 1) Utilizing DISA ACL – Original**



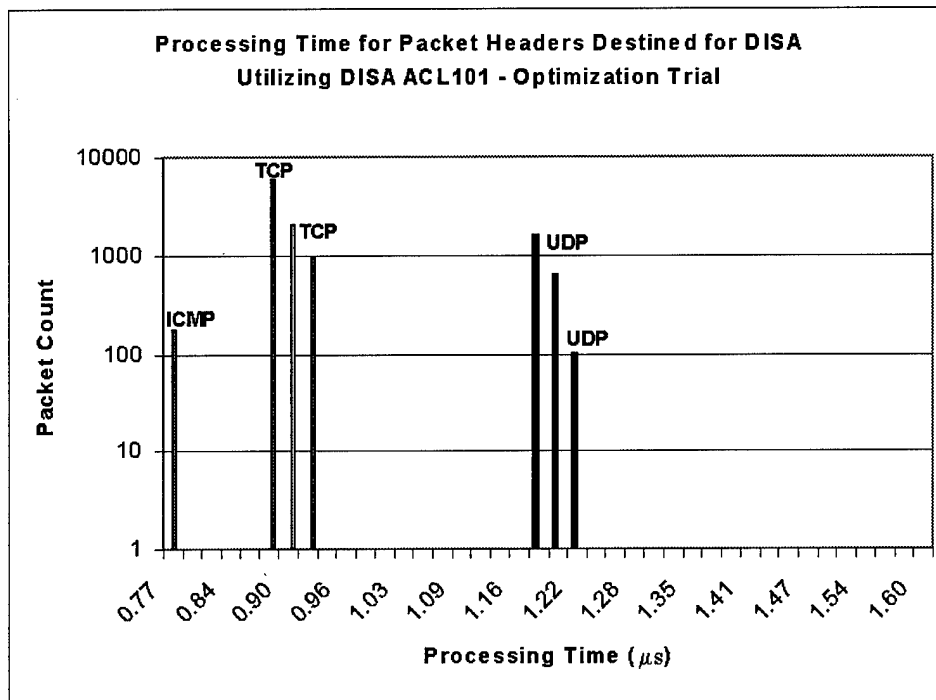
**Figure 44. PT for Random Packets (Seed 2) Utilizing DISA ACL – Original**



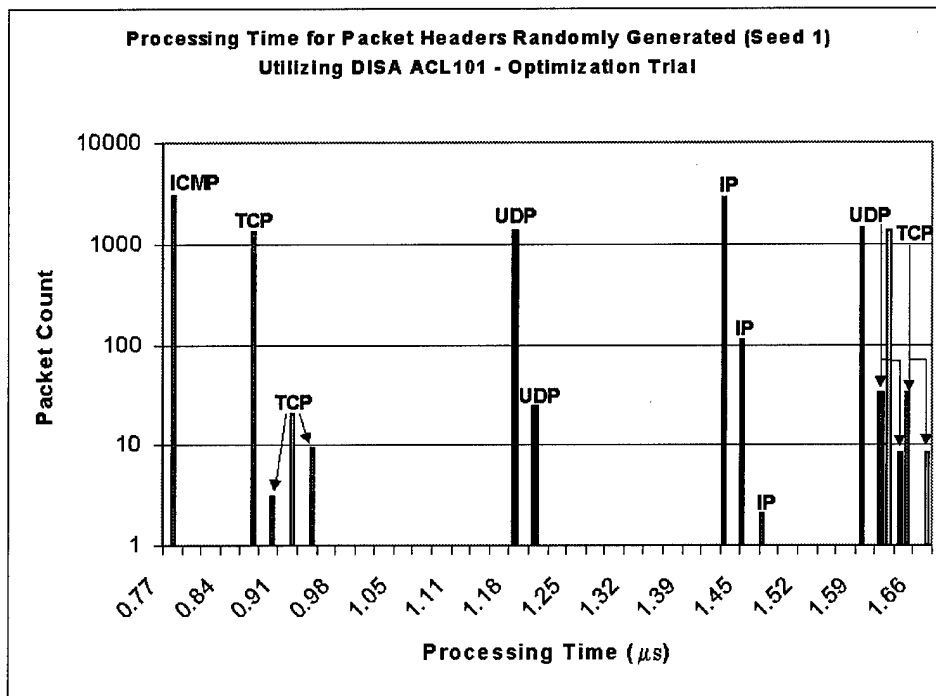
**Figure 45. PT for Random Packets (Seed 3) Utilizing DISA ACL – Original**



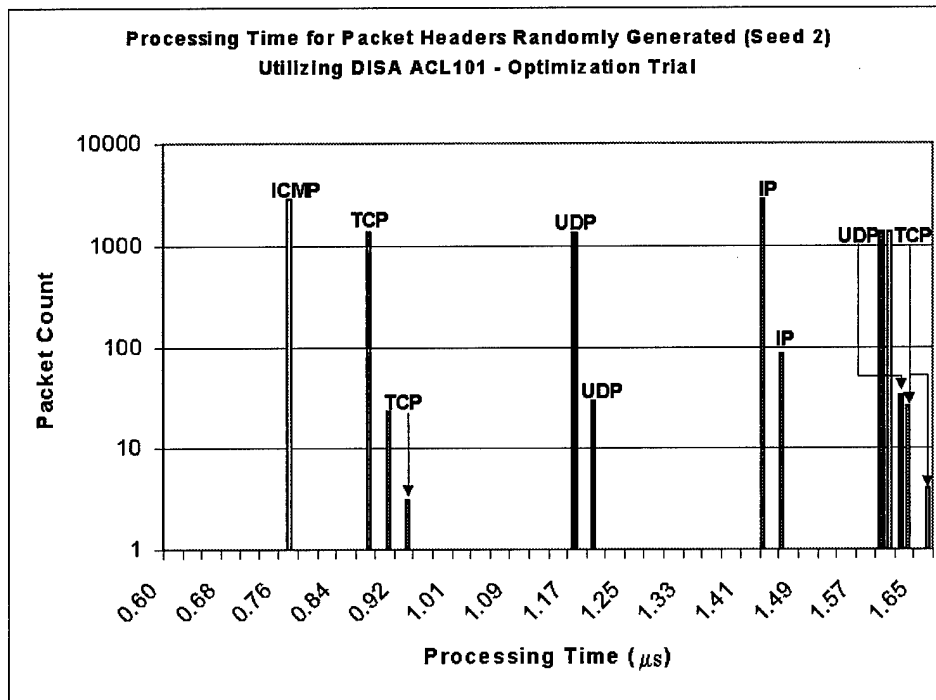
**Figure 46. PT for AFIT Packets Utilizing DISA ACL – Original**



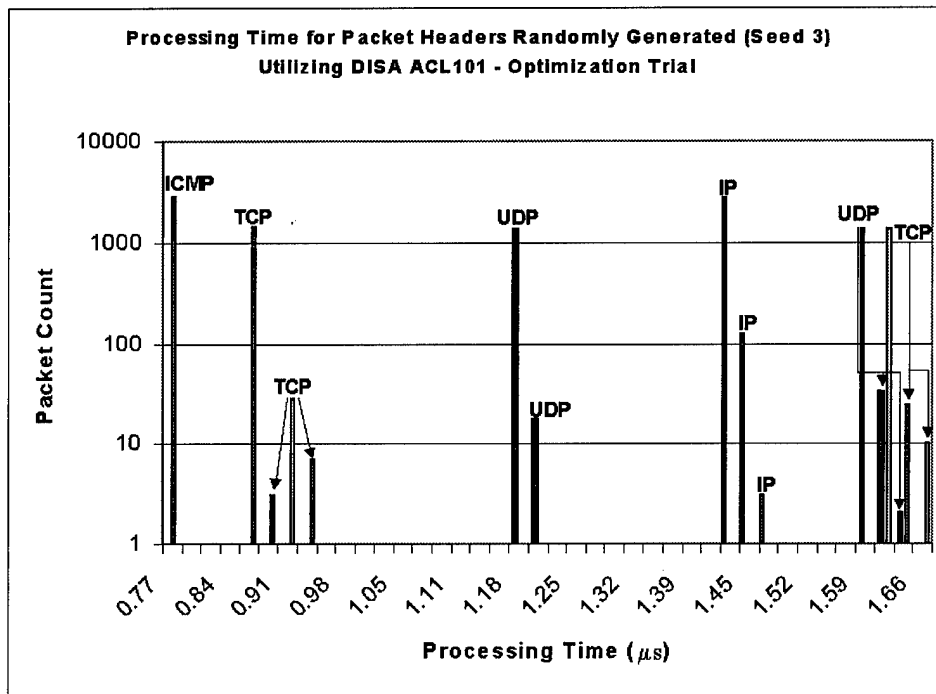
**Figure 47. PT for DISA Packets Utilizing DISA ACL – Optimization Trial**



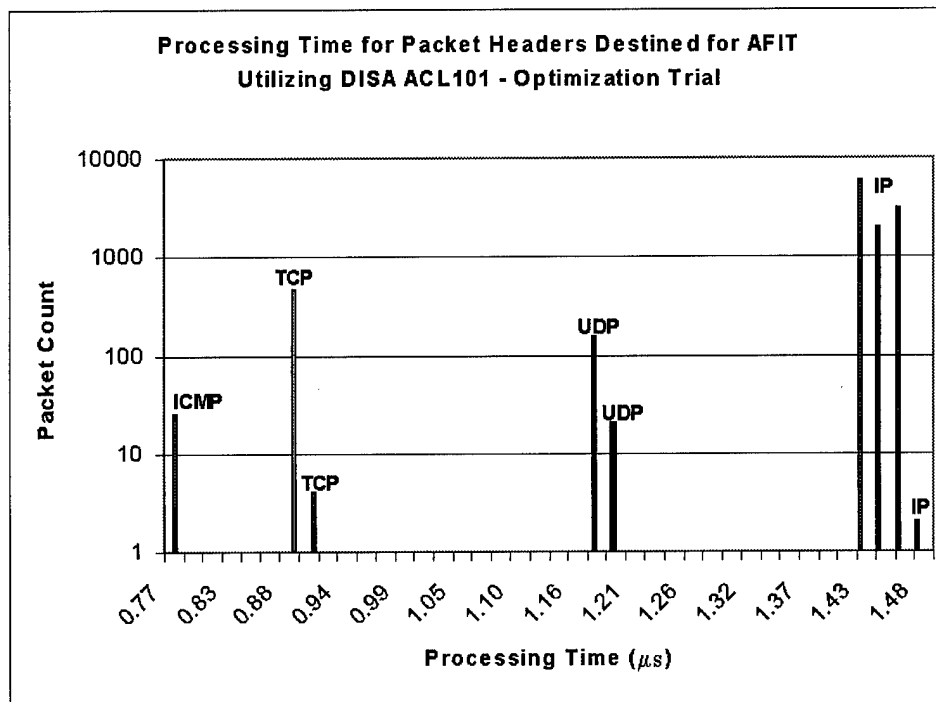
**Figure 48. PT for Random Packets (Seed 1) Utilizing DISA ACL – Optimization Trial**



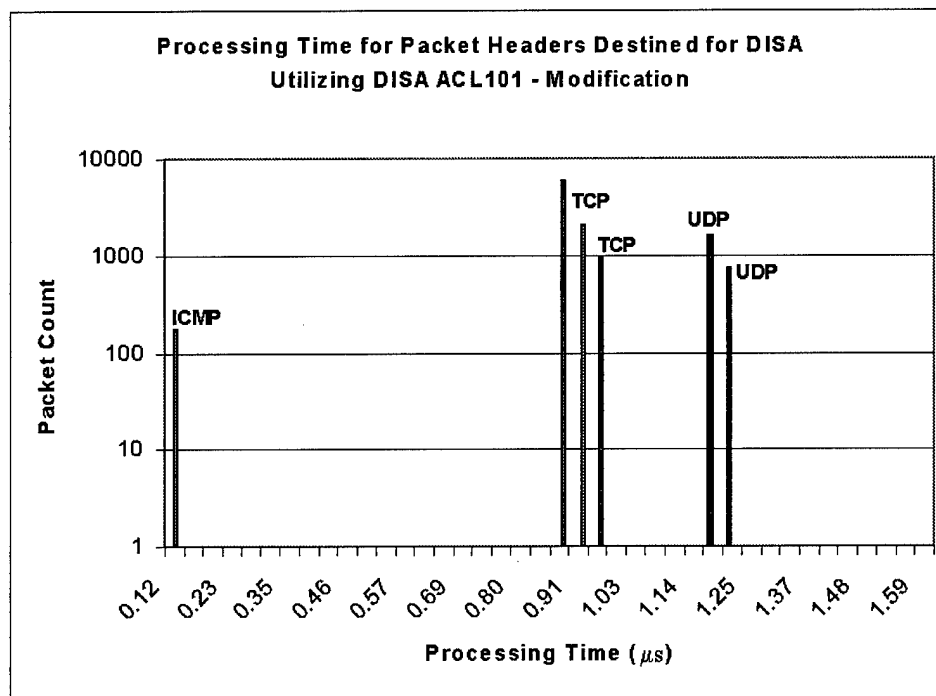
**Figure 49. PT for Random Packets (Seed 2) Utilizing DISA ACL – Optimization Trial**



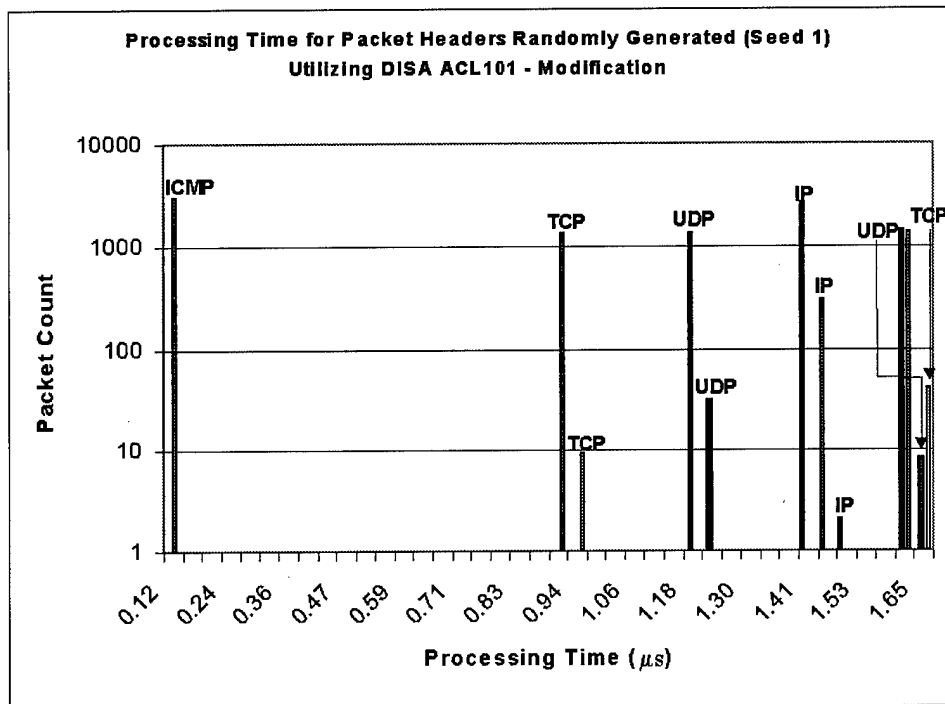
**Figure 50. PT for Random Packets (Seed 3) Utilizing DISA ACL – Optimization Trial**



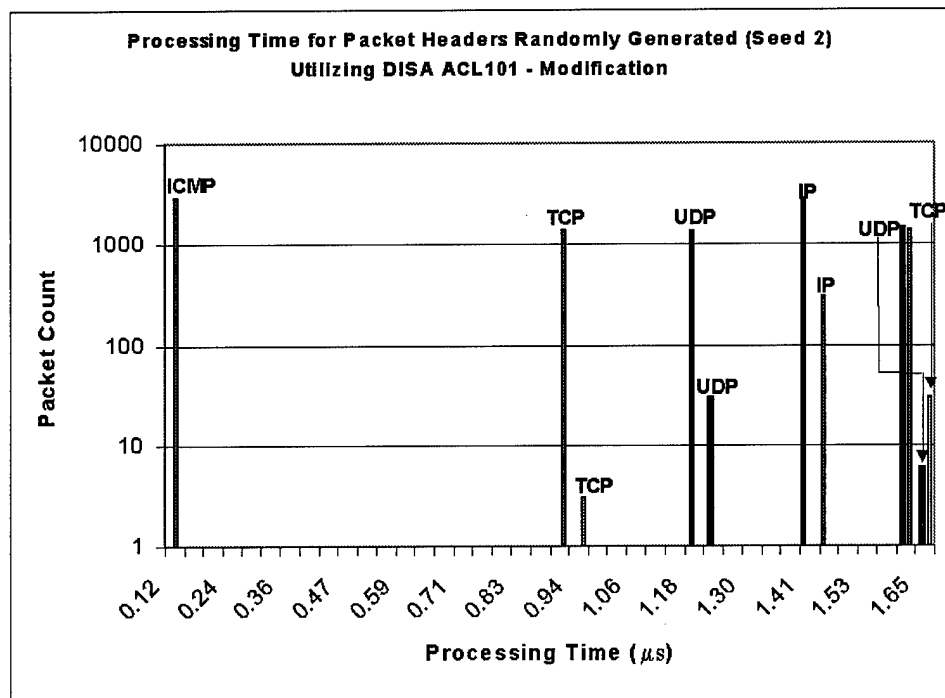
**Figure 51. PT for AFIT Packets Utilizing DISA ACL – Optimization Trial**



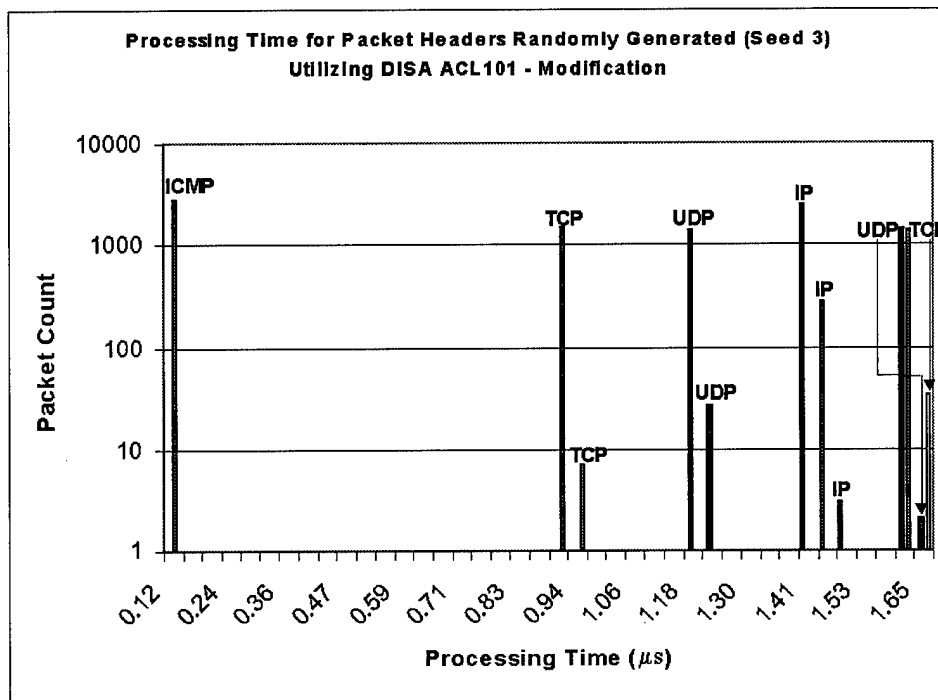
**Figure 52. PT for DISA Packets Utilizing DISA ACL – Modification**



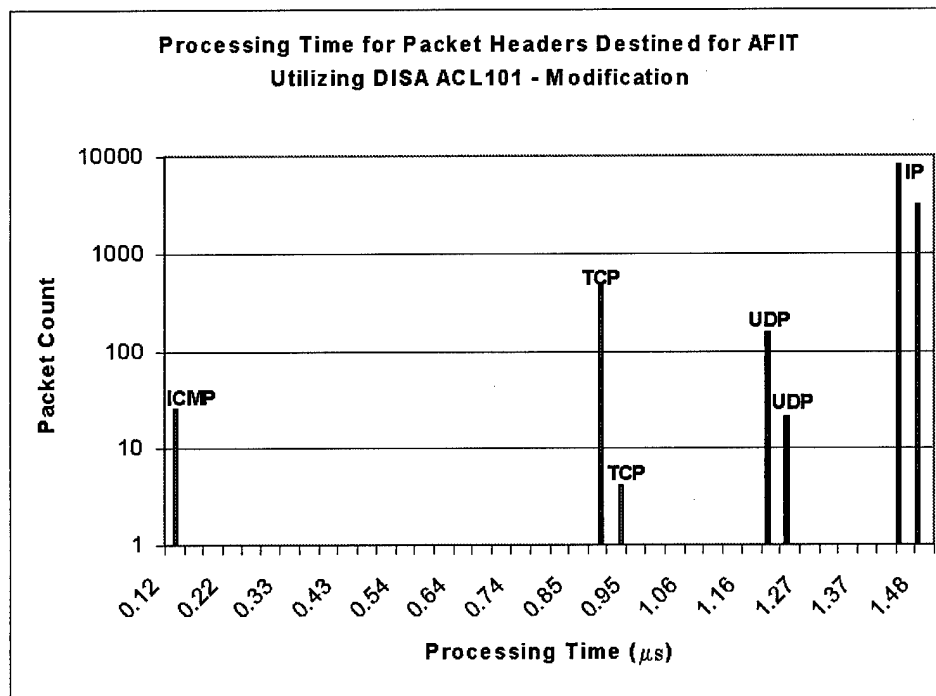
**Figure 53. PT for Random Packets (Seed 1) Utilizing DISA ACL – Modification**



**Figure 54. PT for Random Packets (Seed 2) Utilizing DISA ACL – Modification**



**Figure 55. PT for Random Packets (Seed 3) Utilizing DISA ACL – Modification**



**Figure 56. PT for AFIT Packets Utilizing DISA ACL – Modification**

## Bibliography

- [Bel95] Bellovin, Steve. "Comments on the Nature of the Attack." WWWeb, <http://www.cisco.com/warp/public/701/29.html> (29 July 1995).
- [Cam98] Campbell, Matthew. "US at Mercy of Cyber Terrorists," The Sunday Times. (17 May 1998).
- [Cis95] Cisco Press. "Increasing Security on IP Networks." WWWeb, <http://www.cisco.com/warp/public/701/30.html> (31 July 1995).
- [Cis97] Cisco Press, "Access List Summary." WWWeb, [http://www.cisco.com/univercd/cc/td/doc/product/software/ssr83/rpc\\_r/4108.htm](http://www.cisco.com/univercd/cc/td/doc/product/software/ssr83/rpc_r/4108.htm) (1997).
- [Cis98a] Cisco Press, "Access Control Lists: Overview and Guidelines." WWWeb, [http://www.cisco.com/univercd/cc/td/doc/product/software/ios113ed/113ed\\_cr/secur\\_c/scprt3/scacls.htm](http://www.cisco.com/univercd/cc/td/doc/product/software/ios113ed/113ed_cr/secur_c/scprt3/scacls.htm) (1998).
- [Cis98b] Cisco Press, "Configuring IP Session Filtering (Reflexive Access Lists)." WWWeb, [http://www.cisco.com/univercd/cc/td/doc/product/software/ios113ed/113ed\\_cr/secur\\_c/scprt3/screflex.htm](http://www.cisco.com/univercd/cc/td/doc/product/software/ios113ed/113ed_cr/secur_c/scprt3/screflex.htm) (1998).
- [Cis98c] Cisco Press, "Configuring Lock-and-Key Security (Dynamic Access Lists)." WWWeb, [http://www.cisco.com/univercd/cc/td/doc/product/software/ios113ed/113ed\\_cr/secur\\_c/scprt3/sclock.htm](http://www.cisco.com/univercd/cc/td/doc/product/software/ios113ed/113ed_cr/secur_c/scprt3/sclock.htm) (1998).
- [Cis98d] Cisco Press. "IPSec." WWWeb, [http://www.cisco.com/warp/public/732/Security/ipsec\\_wp.htm](http://www.cisco.com/warp/public/732/Security/ipsec_wp.htm) (29 September 1998).
- [Cis98e] Cisco Press, "Cisco Product Catalog: Chapter 11 – Cisco 7500 Series Product Overview," pages 11-18, Cisco Press. (June 1998).
- [CNN98] Cable New Network. "FBI Opens High-Tech Crisis Center." CNN Interactive WWWeb, <http://cnn.com/us/9811/20/fbi.crisis.center.01/> (20 November 1998).
- [Dae96] Daemon, and others. "IP Spoofing." Phrack Magazine: vol. 7, no. 48, n. pag. WWWeb, <http://www.ozemail.com.au/~geisha/security6.txt> (June 1996).
- [Dav88] Davidson, John. *An Introduction to TCP/IP*. Springer/Verlag : New York, 1988.
- [EsR91] Esponda, Margarita and Ra'ul Rojas. "The RISC Concept – A Survey of Implementations." WWWeb, <http://www.inf.fu-berlin/lehre/WS94/RISC-9.html> (September 1991).
- [Har98] Harris, Phillip. "Your Routers and Their Performance." Packet™ WWWeb, <http://www.cisco.com/warp/public/784/packet/oct98/4.html> (October 1998).
- [HeP94] Hennesey, John L. and David A. Patterson. *Computer Organization & Design The Hardware/Software Interface*. Morgan Kaufmann Publishers, Inc : New York, 1994.



- [Hue97] Huegen, Craig A. "The Latest in Denial of Service Attacks: "Smurfing." WWWeb, <http://www.quadrunner.com/~chuegen/smurf.txt> (October 1997).
- [Kum97] Kumar, G. Prem and Venkataram P. "Security Management Architecture for Access Control to Network Resources," IEE Proceedings – Computer Digital Technology. Vol. 144, No. 6, 1997.
- [Mor98] Morrissey, Peter. "Demystifying Cisco Access Control Lists," Network Computing: vol. 9, no. 7, pg. 116-120 15 April 1998.
- [Nat97] National Computer Security Association. "2<sup>nd</sup> Annual; Firewall Buyer's Guide," 1997.
- [Nyg98] Nygard, Joe. System Engineer, Cisco Systems, San Jose, CA. Personal Correspondence. 13 October 1998.
- [Nyg99] Nygard, Joe. System Engineer, Cisco Systems, San Jose, CA. Personal Correspondence. 2 February 1999.
- [Ran96] Ranum, Marcus J. "ICMP Bombing and IP Slicing." WWWeb, <http://www.clark.net/pub/mjr/pubs/attck/> (1996).

## **Vita**

Captain Douglas R. Lomsdalen was born on 1 January 1968 in Davenport, IA. He graduated from Central Community High School, DeWitt, IA in May 1986. Capt Lomsdalen enlisted in the Air Force on 4 September 1986 and immediately began working on his undergraduate degree; after frequent interruptions he earned a Bachelor of Science degree in Computer Systems from the University of Maryland – European Division in Heidelberg, Germany in May 1993. Capt Lomsdalen completed Officer Training School and was commissioned on 30 September 1994.

Capt Lomsdalen married the former Tatjana Wenglowksi of Zweibrücken, Germany, on 12 November 1988. He has twin sons, Marcus and Jason.

As an officer, Capt Lomsdalen has been assigned to the United States Strategic Command, Offutt AFB, Omaha, NE from October 1994 – August 1997. While there he performed duties related to the maintenance of software used in the allocation of nuclear weapons to targets in the Single Integrated Operation Plan. In August 1997, he entered the Graduate School of Engineering, at the Air Force Institute of Technology. Upon graduation, Capt Lomsdalen will move on to an assignment at The College of Aerospace Doctrine, Research, and Education (CADRE), where he will be designing and maintaining software for the Air Force Wargaming Institute.